ELSEVIER

Artificial Intelligence 90 (1997) 25-77

Artificial Intelligence

Continuous case-based reasoning

A. Ram [1], J.C. Santamaría *

College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA

Received December 1994; revised September 1996

Abstract

Case-based reasoning systems have traditionally been used to perform high-level reasoning in problem domains that can be adequately described using discrete, symbolic representations. However, many real-world problem domains, such as autonomous robotic navigation, are better characterized using continuous representations. Such problem domains also require continuous performance, such as on-line sensorimotor interaction with the environment, and continuous adaptation and learning during the performance task. This article introduces a new method for *continuous case-based reasoning*, and discusses its application to the dynamic selection, modification, and acquisition of robot behaviors in an autonomous navigation system, SINS (self-improving navigation system). The computer program and the underlying method are systematically evaluated through statistical analysis of results from several empirical studies. The article concludes with a general discussion of case-based reasoning issues addressed by this research.

Keywords: Case-based reasoning; Machine learning; Reinforcement learning; Robot navigation; Reactive control; Motor schema-based navigation

1. Introduction

Case-based reasoning systems have traditionally been used to perform high-level reasoning in problem domains that can be adequately described using discrete, symbolic representations. For example, CHEF uses case-based planning to create recipes [21], AQUA uses case-based explanation to understand newspaper stories [45], HYPO uses case-based interpretation for legal argumentation [5], MEDIATOR uses case-based problem solving for dispute resolution [26], and PRODIGY uses case-based reasoning in the form of derivational analogy for high-level robot planning [60].

* Corresponding author. E-mail: carlos@cc.gatech.edu. URL: http://www.cc.gatech.edu/ai/students/jcs.
[1] E-mail: ashwin@cc.gatech.edu. URL: http://www.cc.gatech.edu/faculty/ashwin.

PII S0004-3702(96)00037-9

In our research, we have been investigating the problem of performance and learning in continuous, real-time problem domains, such as autonomous robotic navigation. Continuous problem domains require different underlying representations and place additional constraints on the problem solving process [13]. In this article, we present a new method for case-based reasoning which can be used to guide action and to learn in continuous problem domains. In addition to addressing this class of problems, the research presented here has implications for the design of case-based reasoning systems in general.

Our method, called *continuous case-based reasoning*, shares many of the fundamental assumptions of what might be called "discrete" case-based reasoning in symbolic problem domains.[2] Learning is integrated with performance. Performance is guided by previous experience. New problems are solved by retrieving cases and adapting them. New cases are learned by evaluating proposed solutions and testing them on a real or simulated world. The basic problem solving mechanism relies on the retrieve-adapt-apply-learn cycle common to case-based reasoning systems [21,27,49].

However, the requirements of continuous problem domains are significantly different in ways that do not permit ready application of traditional case-based reasoning methods. For example, consider the problem of driving a car on a highway. Car driving experiences can vary from one another in infinitely many ways. The speed of a car might be 55 mph in one experience and 54 mph in another. Within a given episode, the speed of the car might continuously vary, both infinitesimally from moment to moment, and significantly from, say, the highway to an exit ramp. The problem solving and learning process must operate continuously; there is no time to stop and think, nor a logical point in the process at which to do so.

Such problem domains are "continuous" in three senses. First, they require *continuous representations*. For example, a robotic navigation task requires representations of perceptual and motor control information. The input is a continuous stream of perceptual data from ultrasonic and other sensors; the data itself is analog in the sense that the value of an input parameter can vary infinitesimally (within the limits of the digitization and sampling parameters). Second, continuous problem domains require *continuous performance*. For example, driving a car requires continuous, on-line action. Often, problem solving performance is incremental of necessity because of limited knowledge available to the reasoning system and/or because of the unpredictability of the environment; the system can at best execute the "best" short-term actions available to it and then reevaluate its progress. A robot, for example, may not know where obstacles lie until it actually encounters them. Third, these problem domains require *continuous adaptation and learning*. As the problems encountered become more varied and difficult, it becomes necessary to use fine-grained, detailed knowledge in an incremental manner to act, and to rely on continuous feedback from the environment to adapt actions and learn from experiences.

[2] We do not eschew symbolic representations; rather, the issue is the continuous, time varying nature of any proposed representations, whether symbolic, numeric, or otherwise.

Case-based reasoning in such problem domains requires significant enhancements to the basic case-based reasoning methods used in discrete, symbolic reasoning systems. Several issues need to be addressed. When are two experiences different enough to warrant consideration as independent cases? What is the scope of a single case? For example, is the entire car trip from one's house to the grocery store a single case that can be used to guide and improve one's driving performance in future situations? How should "continuous cases" be represented? How should they be matched and retrieved? How can they be used to guide performance? How are they learned and modified through experience? And how can this performance and learning be integrated into a continuous, on-line, real-time process?

In this article, we provide an answer to these questions based on our research into a robot navigation task. The proposed methods are fully implemented in a computer system which uses reactive control for its performance element and case-based reasoning for continuous adaptation of the performance element and for continuous learning through experience. The computer system, SINS (self-improving navigation system), is implemented on a Denning MRV-III robot. We begin with a description of the task domain, after which we discuss the relevant technical details of the proposed method and the computer system that implements it. We then present several empirical studies of the system to evaluate our approach. The results of these studies are analyzed using statistical methods to demonstrate the efficacy of the approach, to understand the behavior of the system in terms of the design of the computational model, to analyze the impact of different representational and algorithmic choices, to determine the appropriate settings for the system's design parameters under various conditions, to predict system behavior under changing environmental circumstances, and to analyze the "sources of power" behind the proposed methods.

We conclude with the contributions of our research and their implication for the design of case-based reasoning systems in general. We discuss the assumptions underlying our approach, which are common to many case-based reasoning systems. Next, we discuss the merits of fine-grained representations and show how such representations can support on-line learning and adaptation. We discuss the differences between "solution adaptation"—adaptation of the solution recommended by a case to fit a new situation, as in standard case-based reasoning systems—and "case modification"—retroactive modification of a case in response to a situation in which it is used. We introduce the notion of a "virtual case"—a representative experience that the system may or may not have actually had—and show how virtual cases can represent not just past experiences (as in standard case-based reasoning) but also alterations introduced by subsequent similar experiences. Our system, therefore, implements a kind of "continuous dynamic memory", analogous to the dynamic memory of traditional case-based reasoning systems [53]. We argue that these and other innovations introduced in the SINS system, such as time history rather than instantaneous representations, can be beneficial in case-based reasoning systems in general. We also believe that the empirical evaluation methods used here can be used fruitfully in other AI applications not just to evaluate a proposed computer system but also analyze the theoretical model underlying the system.

2. The robot navigation task

Autonomous robotic navigation is defined as the task of finding a path along which a robot can move safely from a source point to a destination point in an obstacle ridden terrain, and executing the actions to carry out the movement in a real or simulated world. Several methods have been proposed for this task, ranging from high-level planning methods to reactive methods. High-level planning methods use extensive world knowledge about available actions and their consequences to formulate a detailed plan before the actions are actually executed in the world (e.g., [15,18,33,51]). Considerable high-level knowledge is also needed to learn from planning experiences (e.g., [21,37,41,55]). Situated or reactive control methods, in contrast, perform no planning in the traditional sense; instead, a simple sensory representation of the environment is used to select the next action that should be performed [3,9,23,43]. Actions are represented as simple behaviors, which can be selected and executed rapidly, often in real time. These methods can cope with unknown and dynamic environmental configurations, but only those that lie within the scope of predetermined behaviors. Furthermore, such methods cannot modify or improve their behaviors through experience, since they do not have any predictive capability that could account for future consequences of their actions, nor a higher-level formalism in which to represent and reason about the knowledge necessary for such analysis.

We have developed a self-improving navigation system that uses reactive control for fast performance, augmented with a continuous case-based reasoning method that allow the system to adapt to novel environments and to learn from its experiences. The system autonomously and progressively constructs representational structures that aid the navigation task by supplying the predictive capability that standard reactive systems lack. The representations are constructed using a hybrid case-based and reinforcement learning method without extensive high-level reasoning. The system is very robust and can perform successfully in (and learn from) novel environments, yet it compares favorably with traditional reactive methods in terms of speed and performance. A further advantage of the method is that the system designers do not need to foresee and represent all the possibilities that might occur since the system develops its own "understanding" of the world and its actions. Through experience, the system is able to adapt to, and perform well in, a wide range of environments without any user intervention or supervisory input. This is a primary characteristic that autonomous agents must have to interact with real-world environments.

Before presenting the technical details of our approach, let us discuss reactive control for robot navigation and some of the approaches that have been proposed to perform the autonomous navigation task.

2.1. Background and related work

Several different architectures for reactive autonomous robotic navigation have been proposed (e.g., [3,9,31,43]). Typically, these methods rely on a combination of several *task achieving* modules, behaviors, or schemas that perform simple substasks such as avoiding obstacles, wandering, or exploration. Each module has a stimulus response type

of relationship with the world. The response of the robot is the result of the interaction of all the responses in the system. The interaction is computed according to different schemes, such as subsumption, in which the response of some modules can supress or subsume the response of other modules (e.g., [9]); weighted summation, in which the final response is a weighted average of individual responses (e.g., [3]); or voting, in which the final response depends on how many modules propose it (e.g., [31]).

One of the main difficulties of the reactive control approach for autonomous navigation is to decide which modules should be active and under what situations. The simplest approach to solve this problem is to precompile the priorities among behaviors manually and keep these priorities constant until the completion of the task (e.g., [3,9]). In this way, the problem of activating modules is avoided altogether since their activation and priorities remain fixed and constant. The problem with this approach is that the system performs well only in specific scenarios that the architecture was designed to solve and those that are "simple" in the sense that one predesigned strategy is sufficient to handle the entire scenario.

A more complex approach for module activation is to precompile "switching circuitry" into the architecture that activates relevant modules under specific conditions (e.g., [4,7,50]). For example, a trash collecting robot might wander looking for an empty can on the floor. During this condition, only the modules avoid obstacle and wander would be active. Once the robot detects a can, it can deactivate the wandering behavior and activate a module responsible for approaching the can. Although this approach can be used to perform more complex navigation tasks, it is still static and predetermined at design time. A more interesting and powerful approach is to let the robot learn which modules should be active as well as the situations that trigger their activation/deactivation. This is the approach pursued in this article. We use continuous case-based reasoning to learn the situations the robot commonly faces as it navigates and to associate a set of parameters to those situations that control not only the activation levels but also the priorities among the modules.

Maes and Brooks [34] propose an algorithm to learn how to coordinate modules in a robot as it navigates. The algorithm uses the robot's experiences to estimate the correlations of module activations and perceptual conditions with positive and negative feedback. The algorithm incrementally updates the estimation of these correlations and activates only those modules that are relevant and reliable to the task, that is, it activates the modules that are positively correlated with positive feedback and negatively correlated with negative feedback under the condition the robot is currently facing. One limitation of this approach is that the robot must decide which modules to activate based only on the currently sensed situation. This becomes a problem when the sensory information is not rich enough to disambiguate between different world situations that are perceived similarly, and thus does not provide enough information to determine which modules to activate and prioritize. For example, a robot that finds itself in a cluttered area may either try to squeeze through or back up. The right choice may depend on past information, such as whether the robot has already tried squeezing through or the density of obstacles the robot will face if it backs up. In our approach, the robot uses not only the currently perceived situation but also the recent history (time sequence) of situations leading up to the current situation.

A less direct approach to behavior selection is proposed by Mataric [35]. Her method uses a functional characterization of the robot's sensors to construct a qualitative map of the environment as the robot navigates. The map is based on landmarks the robot can detect with its sensors. For each navigation task, the robot uses the learned map and the surrounding landmarks to localize itself in the world. Then, it uses the map to plan the best route to the goal. As the robot navigates, it detects new landmarks and verifies its position within the map. Additionally, it selects behaviors that can take it from the current position to the next landmark on the map. However, this and other qualitative map learning approaches (e.g., [29]) rely on hand-coded knowledge about what each behavior or combination of behaviors can achieve when activated using a particular priority scheme. This knowledge enables the robot to decide what modules to activate to accomplish some intermediary objective (e.g., get to the next landmark in the qualitative map). A more robust approach might be to allow the robot to learn not only the disposition of landmarks in the terrain but also knowledge about the effects of its behaviors which can be used to its advantage as it navigates. Another difficulty with the map learning approach is that the acquired knowledge is very dependent on the terrain in which the robot is trained, since it encodes the relative positions of the landmarks in that specific terrain.

2.2. *Motivation for a case-based reasoning approach to behavior coordination*

A robot performing the task of autonomous navigation in complex environments using a reactive control method must be able to coordinate and prioritize the operation of its modules or behaviors. The reason for this is that in complex environments there are many different situations that require different coordination and priorities among motor schemas. For example, in a free space with very few obstacles, the robot should focus on moving towards the goal rather than worry about avoiding obstacles. This would allow the robot to proceed directly towards the goal without any delay. However, in a cluttered space, the robot should pay more attention to avoiding obstacles and slow down its tendency to move towards the goal, since not to do so would invite collisions which are important to avoid. In schema-based reactive control, simple behaviors such as moving towards the goal and avoiding obstacles are implemented by motor schemas, each responsible for one such behavior. Coordination among motor schemas is accomplished by a set of gains or control parameters, which determine the final response of the robot through a weighted summation schema (e.g., [3]). For example, the parameter which modulates the output of the "obstacle avoidance" motor schema affects the robot's tendency to avoid obstacles. In a clear environment with very few obstacles, this parameter should be reduced so as to permit the robot to take a relatively direct path towards the goal.

As mentioned previously, one approach to coordinate motor schema-based behaviors is to precompile their priorities (or gains) at design time (e.g., [4]). In this approach, the designers know the types of situations the robot may encounter and the appropriate coordination scheme to use under those situations. Thus, the robot must simple detect under what situation it is currently navigating and use the precompiled coordination scheme to successfully perform its task. One method for encoding these situations is to use a

library of "cases" which encode navigation strategies for different situations that have been experienced in the past [47]. However, these approaches may fail when the robot encounters situations that were not considered at design time since this knowledge—whether encoded as cases or otherwise—is hand coded by the system designer and not learned or modified by the robot over time. Additionally, designers must determine an appropriate priority scheme for every foreseen situation, which is time consuming and may require the use of an expert, although this process may be automated (e.g., finding optimal control parameters through the use of genetic algorithms as in [46]).

A more robust approach is to provide the robot with only knowledge about which situations result from the execution of specific coordinations of motor schemas and then let the robot to learn the topology of situations in the terrain as it navigates. In this approach, the robot perceives its current situation and selects the specific coordination scheme to achieve either the desired situation, or an intermediate situation that would take it closer to the desired situation. Although not in the context of motor schema-based reactive control, this approach has been explored by other researchers and it is usually refered to as qualitative map learning (e.g., [29,35]).

While useful for specific problems, these and other similar approaches require a priori knowledge in the form of situations and control parameter pairs. Thus, these approaches are not applicable when a priori knowledge is not available, or when such knowledge is not reliable. In such situations, the robot must learn not only the situations that require different coordination schemes for motor schemas but also the appropriate coordination schemes that it should use under those situations. Since a priori knowledge is unavailable or unreliable, the robot must perform this learning based on its own experiences. This suggests an experience-based or case-based approach to this problem: the method should use previous experience to learn relevant situations and appropiate coordination schemes (or control parameters) to use in those situations. Furthermore, since the robot is unlikely to have exactly the right piece of knowledge in memory for a new situation, it must be able to adapt the best candidate to the specifics of the new situation. This again suggests a case-based approach: adaptation is a fundamental part of case-based reasoning [21,27].

Pandya and Hutchinson [42] discuss a case-based approach to robot motion planning. Although this research focusses on motion planning rather than robot navigation, it addresses issues similar to the ones we are exploring. In their approach, a planner has at its disposition a set of path planning strategies or methods that it can use to solve path planning problems. It tries to solve a collision-free path planning problem by retrieving and applying the solution strategies the planner has successfully used to solved previous similar problems. Some of the methods are computationally less expensive than others, but can only solve simple problems. The planner use its own experience to learn when to apply a specific method to a given problem. For every new problem to be solved, the planner retrieves cases with strategies that have been applied to problems with similar descriptions, proposes a solution method, and applies it to the given problem after proper adaptation. The proposed method is tested on a world model to evaluate its effectiveness. In case of a failure, a repairer is used to "diagnose" or "explain" the nature of the failure and to suggest a repaired version of the method, which is then re-tested. The cycle of diagnosis and repair continues until the plan is deemed successful, or until a failure is

encountered that has no clear fix. In any case, the experience is stored as a new case in the case library for future reference.

Another path planning system that uses case-based methods is ROUTER [19]. The system integrates model-based and case-based methods to perform high-level planning of routes that connect any two points in a topological map. The model-based method plans paths by heuristically searching a hierarchical model of the navigation space in which the spatial relationships among occupied and empty regions of such space are represented. The case-based method plans new paths by adapting and combining previously planned paths. The system either selects a specific method to solve a particular problem, or divides the original problem into several, simpler subproblems, recursively solving each subproblem, and combining their solutions. The resulting path is a symbolic description of the path segments a robot may use to navigate to the destination. In ROUTER, the navigation itself is accomplished using traditional (non-case-based) reactive control.

While related to these approaches, our work focusses on case-based reasoning at the robot navigation level, which requires on-line execution and real-time robotic control, rather than at the high-level motion or path planning level, which is typically carried out off line before the plan is actually executed. Additionally, high-level planning relies on a world model that can be used to test and evaluate proposed solutions, whereas the robot navigation task requires interaction with the real world. These differences impose different constraints and demands on the approaches used to solve them.

One such constraint which is central to our problem is that the learning task should be integrated into the performance task: learning should occur as the robot navigates, and navigation should go on to the best of the robot's ability as learning is taking place. This suggests a continuous, on-line case-based reasoning and learning method of the type proposed in this article. In our approach, "cases" consist of time sequences of associations between situations and control parameters. The robot periodically retrieves the case most similar to the current situation and the recent history of situations leading up to it. If the control parameters suggested by the case have been successful in the past, the robot uses the same control parameters, suitably modified, under the current situation. If no case in memory is similar to the current situation, then the robot tries a reasonable set of control parameters (perhaps even picking control parameters at random) and remembers the outcome of this situation in a new case for future use. The following section describes in detail the architecture of the SINS system as well as the methods responsible for selecting and adapting the coordination of behaviors.

3. The SINS system

The self-improving robotic navigation system (SINS) consists of a navigation module, which uses schema-based reactive control methods, and an on-line adaptation and learning module, which uses continuous case-based reasoning supported by reinforcement learning methods. The navigation module is responsible for moving the robot through the environment from the starting location to the desired goal location while avoiding obstacles along the way. The adaptation and learning module has two responsibilities. The adaptation submodule performs on-line adaptation of the reactive control parameters

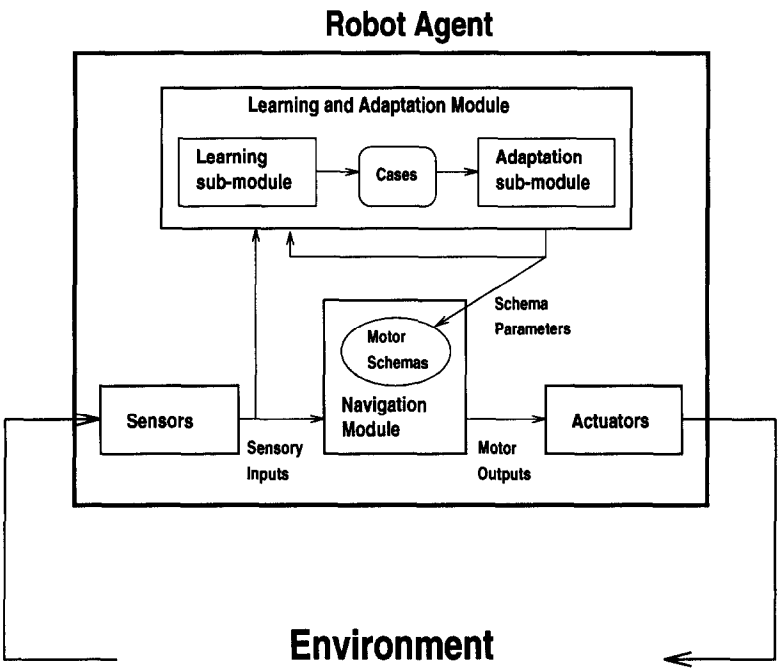

Fig. 1. SINS functional architecture.

to get the best performance from the navigation module. The adaptation is based on recommendations from cases that capture and model the interaction of the system with its environment. With such a model, the system is able to predict future consequences of its actions and act accordingly. The learning submodule monitors the progress of the system and incrementally modifies the case representations through experience. Fig. 1 shows the functional architecture of the system. Note that the learning and adaptation module observes and adapts the navigation module, which in turn drives the robot.

3.1. Schema-based reactive control

The reactive control navigation module is based on the AuRA architecture [3], and consists of a set of motor schemas that represent the individual motor behaviors available to the system. Each schema reacts to sensory information from the environment, and produces a velocity vector representing the direction and speed at which the robot is to move given current environmental conditions. For example, the schema AVOID-STATIC-OBSTACLE directs the system to move itself away from detected obstacles, and the associated schema parameter *Obstacle-Gain* determines the magnitude of the repulsive potential field generated by the obstacles perceived by the system. The velocity vectors produced by all the schemas are then combined to produce a potential field that directs the actual movement of the robot. Simple behaviors, such as wandering, obstacle avoidance, and goal following, can combine to produce complex emergent behaviors in

a particular environment. Different emergent behaviors can be obtained by modifying the simple behaviors.

A detailed description of schema-based reactive control methods can be found in [3]. In this research, we used three motor schemas: AVOID-STATIC-OBSTACLE, MOVE-TO-GOAL, and NOISE. AVOID-STATIC-OBSTACLE directs the system to move itself away from detected obstacles. MOVE-TO-GOAL schema directs the system to move towards a particular point in the terrain. The NOISE schema makes the system move in a random direction; it is used to escape from local minima and, in conjunction with other schemas, to produce wandering behaviors. Each motor schema has a set of parameters that control the potential field generated by the motor schema. In this research, we used the following parameters: *Obstacle-Gain*, associated with AVOID-STATIC-OBSTACLE, determines the magnitude of the repulsive potential field generated by the obstacles perceived by the system; *Goal-Gain*, associated with MOVE-TO-GOAL, determines the magnitude of the attractive potential field generated by the goal; *Noise-Gain*, associated with NOISE, determines the magnitude of the noise; and *Noise-Persistence*, also associated with NOISE, determines the duration for which a noise value is allowed to persist.

3.2. Behavior selection and modification

Our first attempt at building a case-based reactive navigation system focussed on the issue of using case-based reasoning to guide reactive control. A central issue here is the nature of the guidance: at what grain size should the reactive control module represent its behaviors, and what kind of "advice" should the case-based reasoning module provide to the reactive control module? To investigate these issues, we built the ACBARR (a case-based reactive robotic) system, a forerunner to SINS, which focussed solely on using case-based reasoning to guide reactive control but did not deal with the issue of learning the cases necessary to do so.

In order to achieve more robust robotic control, ACBARR used sets of behaviors, called *behavior assemblages*, to represent appropriate collections of cooperating behaviors for complex environments; *behavior switching* to dynamically select behaviors appropriate for a given environment; and *behavior adaptation* to adapt and fine tune existing behaviors dynamically in novel environments [47]. There are two types of behavior adaptations that might be considered. One option is to have the system modify its current behaviors based on immediate past experience. This is a local response to the problem. A more global solution is to have the system select completely new assemblages of behaviors based on the current environment in which it finds itself. A robust system should be able to learn about and adapt to its environment dynamically in both these ways.

Different combinations of schema parameters produce different behaviors to be exhibited by the system (see Fig. 2). This allows the system to interact successfully in different environmental configurations requiring different navigational "strategies". Traditionally, parameters are fixed and determined ahead of time by the system designer. However, on-line selection and modification of the appropriate parameters based on the current environment can enhance navigational performance. We tested this idea by evaluating ACBARR qualitatively and quantitatively through extensive simulation stud-

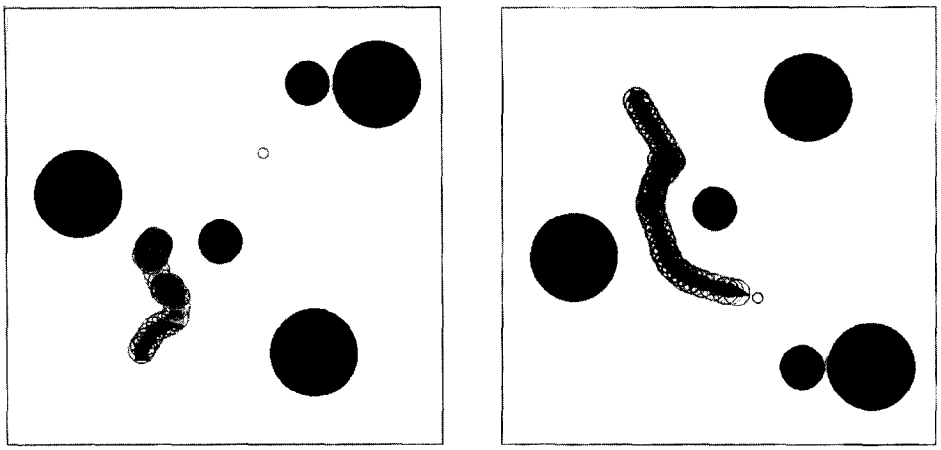

Fig. 2. Typical navigational behaviors of different tunings of the reactive control module. The figure on the left shows the non-learning system with high obstacle avoidance and low goal attraction. On the right, the learning system has lowered obstacle avoidance and increased goal attraction, allowing it to "squeeze" through the obstacles and then take a relatively direct path to the goal (top center).

ies using a variety of different environments and several different performance metrics (see [47], for details). The experiments show that ACBARR is very robust, performing well in novel environments. Additionally, it is able to navigate through several "hard" environments, such as box canyons, in which traditional reactive systems would perform poorly.

In the ACBARR system, we incorporated both behavior adaptation and behavior switching into a reactive control framework. At the local level, this is accomplished by allowing the system to adapt its current behavior in order to build "momentum". If something is working well, the system continues doing it and tries doing it a little bit harder; conversely, if things are not proceeding well, the system attempts something a little different. This technique allows the system to fine tune its current behavior patterns to the exact environment in which it finds itself. For example, if the robot has been in an open area for a period of time and has not encountered any obstacles, it picks up speed and does not worry as much about obstacles. If, on the other hand, it is in a cluttered area, it lowers its speed and treats obstacles more seriously. For schema-based reactive systems, this translates into altering the schema gains and parameters continuously, provided the system has a method for determining the appropriate modifications. ACBARR uses a case-based reasoning method to retrieve rules for behavior modification. These rules are then used to set gain and parameter values appropriate for the environment encountered by the system, and to alter these values incrementally based on current environmental conditions and past successes.

The other method for behavior modification in ACBARR is at a more global level. If the system is currently acting under the control of an assemblage of behaviors which are no longer suited to the current environment, it selects a new assemblage based on what the environment is now like. Continuing with the above example, suppose that the robot

is in a very cluttered environment and is employing a conservative assemblage of motor behaviors. It then breaks out of the obstacles and enters a large open field (analogous to moving from a forested area into a meadow). If only local changes were allowed, the robot would eventually adjust to the new environment. However, by allowing a global change to take place, the system needs only to realize that it is in a radically new environment and to select a new assemblage of motor behaviors, one better suited to the new surroundings. Interestingly, case-based reasoning is used to realize this type of modification as well.

Assemblages of behaviors are represented as *cases*, or standard scenarios known to the system, that can be used to guide performance in novel situations. As in a traditional case-based reasoning system [21,27,49], a case is used to propose a plan or a solution (here, a behavior assemblage) to the problem (here, the current environmental configuration). However, our method differs from the traditional use of case-based reasoning in an important respect. A case in our system is also used to propose a set of *behavior adaptations*, rather than merely the behaviors themselves. This allows the system to use different strategies in different situations. For example, the system might use a "cautious" strategy in a crowded environment by gradually slowing down and allowing itself to get closer to the surrounding obstacles. In order to permit this, strategies suggest boundaries on behavioral parameters rather than precise values for these parameters. Cases are used both to suggest behavior assemblages as well as to perform dynamic (on-line) adaptation of the parameters of behavior assemblages within the suggested boundaries. The knowledge required for both kinds of suggestions is stored in a case, in contrast with traditional case-based reasoning systems in which cases are used only to suggest solutions, and a separate library of adaptation rules is used to adapt a solution to fit the current problem. Further details can be found in [47].

Two important requirements in ACBARR (and SINS) are the ability to manipulate continuous representations and the ability to perform continuously in real time. Manipulation of continuous representations is required because often it is not easy to extract information from sensors in real time to maintain symbolic representations of the environment, nor is it clear a priori what the "right" vocabulary for symbolic features and concepts ought to be. Continuous performance is required because the system needs to modify its behaviors in an on-line manner as it navigates. Thus, in ACBARR, sensor values are preprocessed to produce real-valued variables that represent the current environmental situation. The current environmental information is used to retrieve an appropriate case and to guide behavior adaptation. To guarantee continuous performance, the adaptation information stored in cases is coded as mathematical formulae that can produce a new set of parameter values in a timely manner.

3.3. Case representation

While ACBARR demonstrated the feasibility of on-line case-based reasoning in systems requiring continuous, real-time response, it relied on a fixed library of cases that were hand coded into the system. The system could adapt to novel environments—an important source of flexibility—but it could not improve its own adaptation behavior through experience. Since the knowledge required for behavior adaptation is stored in

cases, we turned our attention to the problem of learning cases through experience. We built SINS (self-improving navigation system), which is similar to the ACBARR system but can learn and modify its own cases through experience. The representation of the cases in SINS is different and is designed to support learning, but the underlying ideas behind the two systems are very similar.

A primary motivation behind SINS was to avoid relying on hand-coded, high-level domain knowledge. There are several disadvantages of relying on such knowledge. First, it is based on an a priori model of the interaction of the robot and its environment. Such a model is an approximation to reality and may not cover all the relevant aspects for successful performance, especially in novel circumstances not anticipated by the system designer. Second, such a model is based on the designer's understanding of the robot and the environment, and thus the quality of the model depends highly on the knowledge and skills of the system designer. Third, it is unclear how a system could extract the necessary high-level knowledge from low-level sensory input in real time; this is one of the standard motivations for developing reactive systems. Fourth, from a theoretical standpoint, a user-designed high-level representation does not provide a scientific explanation of how such knowledge comes to be learned in the first place.

Thus, the representations used by SINS to model its interaction with the environment are initially under-constrained and generic; they contain very little useful information for the navigation task. As the system interacts with the environment, the learning module gradually modifies the content of the representations until they become useful and provide reliable information for adapting the navigation system to the particular environment at hand.

The learning and navigation modules function in an integrated manner. The learning module is always trying to find a better model of the interaction of the system with its environment so that it can tune the navigation module to perform its function better. The navigation module provides feedback to the learning module so it can build a better model of this interaction. The behavior of the system is then the result of an equilibrium point established by the learning module, which is trying to refine the model, and the environment which is complex and dynamic in nature. This equilibrium may shift and need to be re-established if the environment changes drastically; however, the model is generic enough at any point to be able to deal with a very wide range of environments.

The navigation module in SINS can be adapted to exhibit many different behaviors. SINS improves its performance by learning how and when to tune the navigation module. In this way, the system can use the appropriate behavior in each environmental configuration encountered. The learning module, therefore, must learn about and discriminate between different environments, and associate with each the appropriate adaptations to be performed on the motor schemas. This requires a representational scheme to model, not just the external environment, but also the interaction between the system and the environment. However, to ensure that the system does not get bogged down in extensive high-level reasoning, the knowledge represented in the model must be based on perceptual and motor information easily available at the reactive level.

SINS uses a model consisting of associations between the sensory inputs (e.g., *Obstacle-Density*) and schema parameters values (e.g., *Obstacle-Gain*, associated with the AVOID-STATIC-OBSTACLE schema). Each set of associations is represented as a

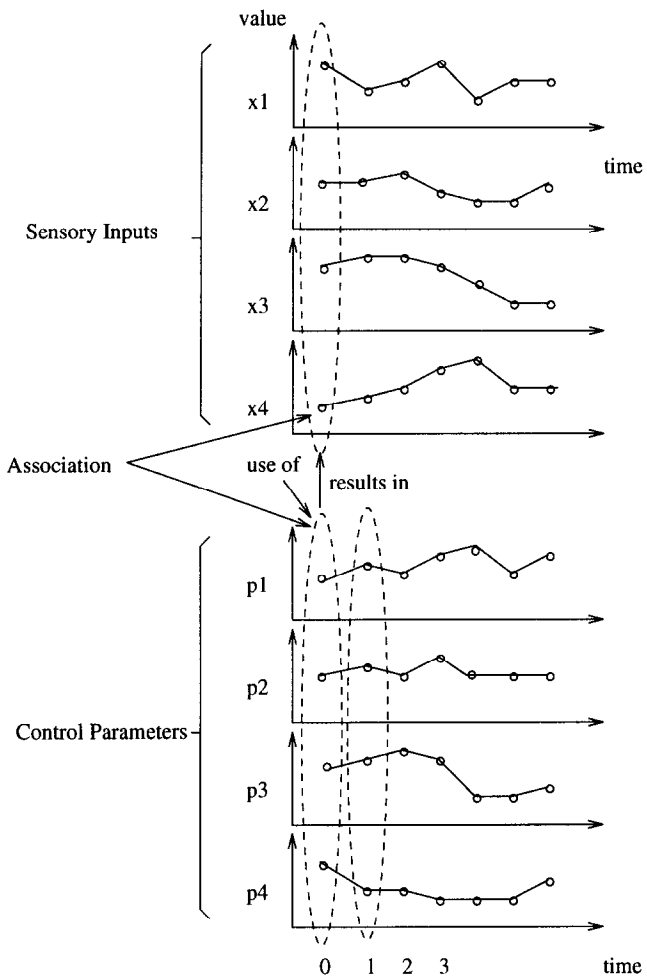

Fig. 3. Sample representations showing the time history of analog values representing perceived inputs and schema parameters. Associations between sensory inputs and control outputs are arranged vertically, and the sequence of associations over time is arranged horizontally. Each case in the system is represented in this manner, as is the current on-going navigational experience of the system.

case. Sensory inputs provide information about the configuration of the environment, and is obtained from the system's sensors. Schema parameter information specifies how to adapt the reactive module in the environments to which the case is applicable. Each type of information is represented as a vector of analog values. Each analog value corresponds to a quantitative variable (a sensory input or a schema parameter) at a specific time. A vector represents the trend or recent history of a variable. A case models an association between sensory inputs and schema parameters by grouping their respective vectors together. Fig. 3 shows an example of this representation.

This representation has three essential properties. First, the representation is capable of capturing a wide range of possible associations between sensory inputs and schema parameters. Second, it permits continuous progressive refinement of the associations. Finally, the representation captures trends or patterns of input and output values over time. This allows the system to detect patterns over larger time windows rather than having to make a decision based only on instantaneous values of perceptual inputs. This ability can be thought of as a kind of "time history clustering"; this is discussed in more detail later.

In this research, we used four input vectors to characterize the environmental and discriminate among different environment configurations: *Obstacle-Density* provides a measure of the occupied areas that impede navigation; *Absolute-Motion* measures the activity of the system; *Relative-Motion* represents the change in motion activity; and *Motion-Towards-Goal* specifies how much progress the system has actually made towards the goal. These input vectors are constantly updated with the information received from the sensors. An important characteristic of these input parameters is that they form a propioceptive representation of the world [2]. That is, the representation is insensitive to isometric translations and rotations of the world configuration. Additionally, the input parameters encode information at a higher level than the information directly perceived.[3] For example, *Obstacle-Density* summarizes how cluttered the environment surrounding the robot is but does not encode the positions of the individual obstacles with respect to the robot. Similarly, *Motion-Towards-Goal* measures the movement of the robot towards the goal but does not encode its location. This coarse representation allows the system to apply similar control parameter values in similar environmental situations and yet discriminate among different situations that would require different control parameter values. Thus, the system's learning is not specific to a given world or goal location; learned navigational strategies are general (such as "when approaching a cluttered area, slow down") and can be deployed in a wide range of situations.

We also used four output vectors to represent the schema parameter values used to adapt the navigation module, one for each of the schema parameters (*Obstacle-Gain*, *Goal-Gain*, *Noise-Gain*, and *Noise-Persistence*) discussed earlier. The values are set periodically according to the recommendations of the case that best matches the current environment. The new values remain constant for a "control interval" until the next setting period.

The choice of input and output vectors was based on the complexity of their calculation and their relevance to the navigation task. The input vectors were chosen to represent environment configurations in a generic manner but taking into account the processing required to produce those vectors (e.g., obstacle density is more generic than obstacle position, and can be obtained easily from the robot's ultrasonic sensors). The output vectors were chosen to represent directly the actions that the learning module uses to tune the navigation module, that is, the schema parameter values themselves.

[3] Section 5 analyzes the impact of this and other design decisions in SINS, including choice of input representation, choice of adaptation method, size of the case library, and so on, via systematic empirical studies.

3.4. Case learning

The case-based reasoning and learning module creates, maintains, and applies the case representations used for on-line adaptation of the reactive module. The main objective of the learning method is to construct a model of the continuous sensorimotor interaction of the system with its environment, that is, a mapping from sensory inputs to appropriate behavioral (schema) parameters. This model allows the adaptation module to continuously control the behavior of the navigation module by selecting and adapting schema parameters in different environments. To learn a mapping in this context is to detect and discriminate among different environment configurations, and to identify the appropriate schema parameter values to be used by the reactive module, in a dynamic and on-line manner. This means that, as the system is navigating, the learning module is perceiving the environment, detecting an environment configuration, and modifying the schema parameters of the reactive module accordingly, while simultaneously updating its own cases to reflect the observed results of the system's actions in various situations.

The method is based on a combination of ideas from case-based reasoning and learning, which deals with the issue of using past experiences to deal with and learn from novel situations, and from reinforcement learning, which deals with the issue of updating the content of system's knowledge based on feedback from the environment (see [57]). However, in traditional case-based planning systems (e.g., [21]) learning and adaptation requires a detailed model of the domain. This is exactly what reactive planning systems are trying to avoid. Earlier attempts to combine reactive control with classical planning systems (e.g., [10]) or explanation-based learning systems (e.g., [39]) also relied on deep reasoning and were typically too slow for the fast, reflexive behavior required in reactive control systems. Unlike these approaches, our method does not fall back on slow non-reactive techniques for improving reactive control.

Each case represents an observed regularity between a particular environmental configuration and the effects of different actions, and prescribes the values of the schema parameters that are most appropriate (as far as the system knows based on its previous experience) for that environment. The learning module performs the following tasks in a cyclic manner:

(1) *perceive* and represent the current environment;
(2) *retrieve* a case whose sensory input vector represents an environment most similar to the current environment;
(3) *adapt* the schema parameter values in use by the reactive control module by installing the values recommended by schema parameter vectors of the case; and
(4) *learn* new associations and/or adapt existing associations represented in the case to reflect any new information gained through the use of the case in the new situation to enhance the reliability of their predictions.

In traditional case-based reasoning terms (e.g., [27]), these steps correspond to situation assessment, case retrieval, adaptation, and learning, respectively; critiquing and evaluation are carried out by executing the behaviors suggested by the case (i.e., modifying the control parameters and carrying out the suggested movements) and observing the actual effects using sensory information.

The *perceive* step builds a set of vectors representing the sensory input, which are then matched against the corresponding vectors of the cases in the system's memory in the *retrieve* step. The case similarity metric is based on the mean squared difference (i.e., Euclidean distance) between each of the vector values of the case over a trending window and the vector values of the environment. The best match window is calculated using a reverse sweep over the time axis similar to a convolution process to find the relative position that matches best. The best matching case is handed to the *adapt* step, which selects the schema parameter values from the case and modifies the corresponding values of the reactive behaviors currently in use using the case similarity metric and a scalar reward. (The reward signal corresponds to the notion of "outcome" in traditional case-based reasoning terms; it is used to reinforce good performance and is discussed in more detail later.) Thus, the actual adaptations performed depend both on the goodness of match between the case and the environment and the usefulness of the chosen parameters as indicated by the reward signal. Intuitively, as long as the reward signal indicates that the performance is satisfactory, a better match between the current environment situation and the retrieved case will result in a higher probability that the system will use the parameters suggested by the case. This can be thought of as a kind of basic animal reinforcement learning or "law of effect", which is based on the common idea of increasing or decreasing the probability of executing an action in a particular state if experience shows that such action is beneficial or prejudicial, respectively [59]; however, unlike traditional stimulus response learning and like in most AI learning systems, learning in SINS is mediated by intermediate representations (cases).

Finally, the *learn* step uses statistical information about prior applications of the case to determine whether information from the current application of the case should be used to modify this case, or whether a new case should be created. The vectors encoded in the cases are adapted using a reinforcement formula in which a *relative similarity measure* is used as a scalar reward or reinforcement signal. The relative similarity measure quantifies how similar the current environment configuration is to the environment configuration encoded by the case relative to how similar the environment has been in previous utilizations of the case. Intuitively, if case matches the current situation better than previous situations it was used in, it is likely that the situation involves the very regularities that the case is beginning to capture; thus, it is worthwhile modifying the case in the direction of the current situation. Alternatively, if the match is not quite as good, the case should not be modified because that will take it away from the regularity it was converging towards. Finally, if the current situation is a very bad fit to the case, it makes more sense to create a new case to represent what is probably a new class of situations.

In summary, as the system navigates it encounters new situations and schema parameters are tuned so that the system can adjust its behavior accordingly. The system learns by remembering which situations it has faced in the past and which control parameters have produced good outcomes during those situations. The system uses two measures of success to provide feedback to the learning algorithm. First, the relative similarity metric ensures that cases capture consistent associations between situations and control parameter values. This increases the likelihood that control parameters produce similar

results in future situations. Second, the external reward signal ensures that appropriate control parameters are associated with each situation. In this way, the system can learn control parameters that produce positive outcomes as measured by the reward signal.

A detailed description of each step is presented below. Note that since case modification is performed using a reinforcement formula based on a relative similarity measure, the overall effect of the learning process is to cause the cases to converge on stable associations between environment configurations and schema parameters. Stable associations represent regularities in the world that have been identified by the system through experience, and provide the predictive power necessary to navigate in future situations. The assumption behind this method is that the interaction between the system and the environment can be characterized by a finite set of causal patterns or associations between the sensory inputs and the actions performed by the system. The method allows the system to learn these causal patterns and to use them to modify its actions by updating its schema parameters as appropriate.

4. Technical details

The continuous case-based reasoning algorithm, as implemented in SINS, is as follows:

Algorithm 1 (*SINS algorithm*).

```
do
  current_environment = perceive();
  if (end of control interval) then
  {
    if (outcome was good) then
    {
      reinforce_schemas(previous_case, current_environment);
      if (prediction is good and at end of sequence) then
        extend_case(previous_case);
    }
    else
      explore_schemas(previous_case);
    best_case = retrieve_best_case(current_environment);
    if (best_case is not a good match) then
      best_case = create_case(current_environment);
    adapt_schemas(best_case);
    previous_case = best_case;
  }
  execute()
while (not (goal reached or maximum number of steps exceeded));
```

The *perceive* function constructs and maintains a representation of the current environmental situation by reading the robot's sensors and updating the input and output vectors accordingly. Recall that these representations are descriptions, not of the current instant in time, but of the sequence of values of the parameters over a given time interval. Thus, the *perceive* function results in a set of $J = 4$ input vectors $\boldsymbol{E}_{\text{input}_j}$, one for each sensory input j, and $K = 4$ output vectors $\boldsymbol{E}_{\text{output}_k}$, one for each output vector k as described earlier. Then, on every control interval T, the learning and adaptation module performs two main functions: it adapts the schema parameters currently in use by the reactive control module so that it performs better in the new environment, and it learns useful sequences of associations between environment situations and schema parameters.

Schema parameters are adapted using the *retrieve_best_case* and *adapt_schemas* functions. In the *retrieve_best_case* function, the case most similar to the current environment situation is selected by matching the environment's input and output vectors $\boldsymbol{E}_{\text{input}_j}$, $\boldsymbol{E}_{\text{output}_k}$ from the *perceive* step against the corresponding input and output vectors $\boldsymbol{C}^n_{\text{input}_j}$, $\boldsymbol{C}^n_{\text{output}_k}$ of the cases $\boldsymbol{C}^n$ in the system's memory (see Fig. 4). The best matching case $\boldsymbol{C}^{n_{\text{best}}}$ and the position of the best match p_{best} are handed to the *adapt_schemas* function, which modifies the schema parameter values currently in use based on the recommendations $\boldsymbol{C}^{n_{\text{best}}}_{\text{output}_j}(p_{\text{best}} + 1)$ from the output vectors of the case.

Finally, the learning and adaptation module decides how to utilize information from the current experience with the best case in order to improve its case library. The system learns in three different ways: by improving the content of the case that was just used in order to make it more reliable, by creating a new case whenever the best case retrieved is not good enough, or by extending the length of the case in order to build up longer sequences of associations. The contents of a case are improved by the *reinforce_schemas* function, which reinforces the suggestions of the case if these suggestions led to a favorable outcome over the last control interval, and by the *explore_schemas* function, which uses random exploration to try out other schema parameter values if the suggested set of values did not prove useful. Random exploration is used because the main objective is to find parameter values that increase the likehood of obtaining a positive outcome in the next cycle. However, in order to reduce the probability of selecting a poor set of parameter values, the selection of parameter values is done non-uniformly, that is, values that have produced positive outcomes (or obtained positive rewards) are more likely to be selected than those that have produced negative outcomes. This is implemented using a non-uniform probability distribution for the random exploration function.[4] The outcome is evaluated by monitoring the behavior of the robot over the last control interval. In our application, collisions are undesirable, as is lack of movement; in other applications, any other suitable performance metric could be used instead.

In traditional case-based reasoning systems, case adaptation is carried out using a rule-based system which utilizes a hand-coded set of adaptation rules (e.g., [24] but cf. [32]). SINS, in contrast, uses a kind of reinforcement learning method to provide

[4] Section 5.7 presents two alternatives implemented in the SINS system and evaluated through systematic empirical studies.

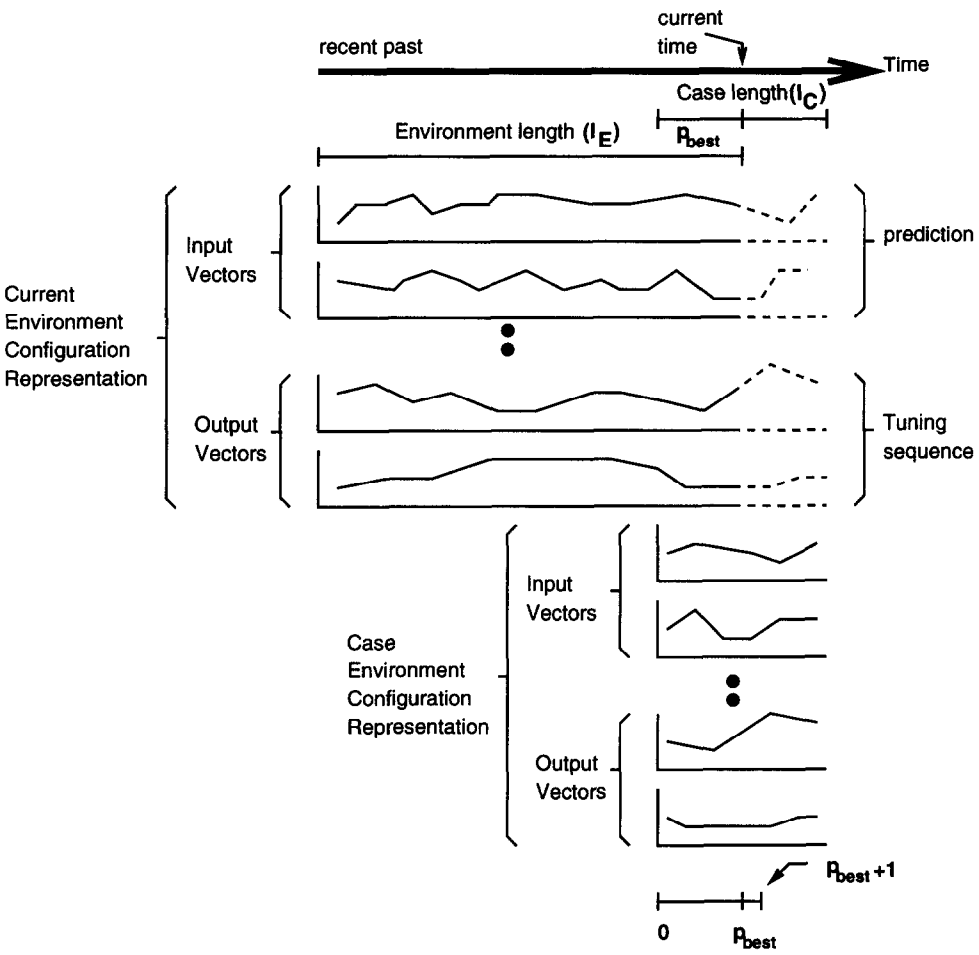

Fig. 4. Schematic representation of the match process. Each graph in the case (below) is matched against the corresponding graph in the current environment (above) to determine the best match, after which the remaining part of the case is used to guide navigation (shown as dashed lines).

the functionality necessary for case adaptation. The issues underlying the integration of multiple learning strategies into a single multistrategy learning system is discussed in more detail in [48]. One difference between our methods and traditional reinforcement learning is that SINS is trying to maximize consistency in "useful" behaviors as determined by a reward signal, whereas traditional reinforcement learning tries to maximize the expected utility the system is going to receive in the future as determined by the reward signal (cf. [58,62]). In schema-based reactive control navigation, it is inherently a good idea to modify schema parameters in an on-line fashion; however, not all modifications are equally good since some may cause the robot to collide with obstacles or not to move at all. SINS uses the reward signal to decide whether to reinforce a behavior or to explore alternative behaviors; reinforcement, when chosen, is used to

reinforce behaviors that are consistent across experiences. Thus, in addition to external outcome, consistency is used as an "internal" reward signal for the reinforcement learning method.[5]

Furthermore, traditional reinforcement learning assumes a predefined set of known states and actions; it learns how to associate actions to states in order to maximize a reward. In SINS, the states and what corresponds to "actions" (the adaptations to be performed on the navigation module) are not known; part of the learning task is to discover the relevant states (the sequences of environmental situations that are likely to result from a given sequence of adaptations) and, in turn, associate appropriate adaptations to different situations. Thus, SINS is learning a model of its sensorimotor interaction with the environment (represented as a set of cases) at the same time as it is learning to improve its navigational performance through on-line adaptation of its reactive control schemas.

In addition to modifying its cases, SINS can also extend its cases and learn new cases. In order to decide which kind of learning to perform in a given situation, SINS uses a relative similarity criterion to judge the appropriateness of the best matching case in the current situation. This determination is based on statistical information about the quality of match in prior applications of the case as compared to the quality of match in the current situation. If the best matching case is not as similar to the current environment situation as it has been in previous situations, the case is probably inappropriate for this situation; thus, it is better to learn a new case to represent what is probably a new class of situations. If this occurs, SINS uses the *create_new_case* function to create a new case based on the current experience and add it to the case library. To determine whether to create a new case, SINS compares the current match with the mean match plus the standard deviation of the matches over the past utilizations of the case. This ensures that new sequences of associations are created only when the available sequences of associations already captured in the case library do not fit the current environment.

The third kind of learning is carried out by the *extend_case_size* function, which extends the length of a case whenever the best case makes an accurate prediction of the next environment situation and there are no more associations in the sequence. This allows the system to increase the length of the sequence of associations only when it is confident that the sequence of the case accurately predicts how the environment changes if the suggested schema parameters are used. To estimate this confidence, the predicted values are matched with the actual environmental parameters that result; if this match is better than the mean match, the case is extended. Intuitively (as before), if the case predicts the current situation better than it predicted the previous situations that it was used in, it is likely that the current situation involves the very regularities that the case is beginning to capture; thus, it is worthwhile extending the case so as to incorporate the current situation. Alternatively, if the match is not quite as good, the case should

[5] SINS can be run with or without an external reward signal. With a reward signal based on external outcome, SINS learns cases that improve its navigational performance; without one, it sometimes learns cases which do not improve performance but nevertheless are "correct" in that they represent regularities in the perception-action model that is being learned. This issue is discussed in more detail in Section 5.9.

not be modified because doing so would take it away from the regularities it has been converging towards.

Since the reinforcement formulae are based on a relative similarity criterion, the overall effect of the learning process is to cause the cases to converge on stable associations between environment configurations and schema parameters, representing regularities in the system-environment interaction that the reward utility has identified as being useful to learn. These regularities capture consistent and predictive perception-action models, and hence can be used as the basis for modifying the schema parameters in different situations.

Genetic algorithms may also be used to modify schema parameters in a given environment [46]. However, while this approach is useful in the initial design of the navigation system, it cannot change schema parameters on line during navigation when the system faces environments that are significantly different from the environments used in the training phase of the genetic algorithm (but cf. [20]). Another approach to self-organizing adaptive control is that of Verschure, Kröse, and Pfeifer [61], in which a neural network is used to learn how to associate conditional stimulus to unconditional responses. Although their system and ours are both self-improving navigation systems, there is a fundamental difference in how the performance of the navigation task is improved. Their system improves its navigation performance by learning how to incorporate new input data (i.e., conditional stimuli) into an already working navigation system, while SINS improves its navigation performance by learning how to adapt the system (i.e., the navigation module) itself. Our system does not rely on new sensory input, but on patterns or regularities detected in perceived environments. Our learning methods are also similar to Sutton [56], whose system uses a trial-and-error reinforcement learning strategy to develop a world model and to plan optimal routes using the evolving world model. Unlike this system, however, SINS does not need to be trained on the same world many times, nor are the results of its learning specific to a particular world, initial location, or destination location.

We now present a detailed description of and the mathematical formulae used in the perception, matching, adaptation, and learning tasks.

4.1. Perception

The objective of the *perceive* function is to generate an accurate description of the current environment situation. It performs this task by shifting the previous values in each input and output vector one position back in time[6] and then calculating the current values for each input vector $\boldsymbol{E}_{\text{input}_j}(0)$, $j = 1, \ldots, J$ and output vector $\boldsymbol{E}_{\text{output}_k}(0)$, $k = 1, \ldots, K$, where 0 is the current position in time. The current values for the input vectors are based on the robot's sensors, and the current values for the output vectors are just the respective values of the schema parameters suggested in the previous control interval. The vectors are updated at the end of each control interval of time T.

[6] This is implemented using a circular buffer which does not require copying each of the values from one cell to the next.

To update the input vectors, the system monitors the values of the robot's sensors $Sensor_j$ corresponding to each input vector $\boldsymbol{E}_{\text{input}_j}$. The sensors are monitored at each time step over the past control interval; these sensor readings are then averaged to yield the new value for the corresponding input vectors. Thus, the input vectors in the environment representation are updated using the following formula:[7]

$$\boldsymbol{E}_{\text{input}_j}(i) = \begin{cases} \boldsymbol{E}_{\text{input}_j}(i-1), & \text{if } i > 0, \\ \dfrac{\sum_{t=-T}^{t=0} Sensor_j(t)}{T}, & \text{if } i = 0, \end{cases}$$

where $Sensor_j(n)$ is the sensory input that corresponds to the input vector $\boldsymbol{E}_{\text{input}_j}$ (sensed obstacles for *Obstacle-Density*, distance traveled for *Absolute-Motion*, relative position for *Relative-Motion*, and normal relative position for *Motion-Towards-Goal*), and t ranges over each robot step since the last control interval.

4.2. Retrieval and matching

The function *retrieve_best_case* is responsible for selecting a case from the case library that best matches the current environment situation. The case similarity metric is based on the mean squared difference between each of the vector values of the case over a trending window, and the vector values of the environment. The best match window is calculated using a reverse sweep over the time axis p similar to a convolution process to find the relative position that matches best. Each case $\boldsymbol{C}^n$ in the case library is matched against the current environment using exhaustive search, which returns the best matching case $\boldsymbol{C}^{n_{\text{best}}}$ along with the relative position p_{best} of the match (see Fig. 4). After retrieving the best case, the mean and variance of the case's statistical match history are updated; these will be used later to calculate the relative similarity criterion during learning.

The case similarity metric SM of a case $\boldsymbol{C}$ at position p relative to the environment $\boldsymbol{E}$ is a value that indicates the similarity between the sequence of associations encoded in the case to the sequence of associations in the current environment situation starting at position p. The lower the value of the case similarity metric, the more similar the sequences of associations. The case similarity metric formula calculates a weighted sum of the squared difference between the corresponding vectors of the case and the environment. For the SM to be valid, p must lie between 0 and l_C.

$$SM(\boldsymbol{E}, \boldsymbol{C}, p) = \sum_{j=1}^{J} w_j \sum_{i=0}^{\min(p, l_E)} \frac{(\boldsymbol{E}_{\text{input}_j}(i) - \boldsymbol{C}_{\text{input}_j}(p-i))^2}{p+1} + \sum_{k=1}^{K} w_k \sum_{i=0}^{\min(p, l_E)} \frac{(\boldsymbol{E}_{\text{output}_k}(i) - \boldsymbol{C}_{\text{output}_k}(p-i))^2}{p+1}.$$

[7] Note that i counts back in time (i.e., $i = 0$ is the current time and $i > 0$ is the recent past).

For this article, we used $w_j = w_k = 1.0$ (i.e., input and output vectors contribute equally in the similarity metric). The best case is obtained by matching each case C^n in the case library at all the positions p and selecting the pair $(n_{\text{best}}, p_{\text{best}})$ that yields the lowest SM. Formally, this can be expressed as:

$$\{n_{\text{best}}, p_{\text{best}} \mid \min(SM(E, C^n, p)),\ \forall n,\ 0 \leqslant p \leqslant l_{C^n}\}.$$

Each case C maintains a statistical record of the similarity metrics it has produced in the past, which is updated every time the case is retrieved as the best case. The mean ($C_{SM_{\text{mean}}}$) and variance ($C_{SM_{\text{var}}}$) of the case similarity metric as well as the number of times the case has been used (C_{used}) are updated using standard formulae in descriptive statistics:

$$\text{new } C_{SM_{\text{mean}}} = \frac{C_{\text{used}} C_{SM_{\text{mean}}} + SM}{C_{\text{used}} + 1},$$

$$\text{new } C_{SM_{\text{var}}} = \frac{C_{\text{used}} - 1}{C_{\text{used}}} C_{SM_{\text{var}}} + (\text{new } C_{SM_{\text{mean}}} - C_{SM_{\text{mean}}})^2 + \frac{(SM - \text{new } C_{SM_{\text{mean}}})^2}{C_{\text{used}}},$$

$$\text{new } C_{\text{used}} = C_{\text{used}} + 1.$$

4.3. Adaptation

The best matching case $C^{n_{\text{best}}}$ is used to adapt the schema parameter values currently in use by the reactive control module. The values of output vectors for the next association $C^{n_{\text{best}}}_{\text{output}_k}$ after position p_{best} are used to determine the new set of schema parameter values $Parameter_k$ until the next control interval. Since learning tends to reinforce those associations that are consistently observed over several experiences, the new set of schema parameters can be expected to cause the robot to move safely and the next environment configuration that results from the movement can be expected to be the one predicted by the association. Since output vectors directly represent schema parameters, adaptation is a straightforward operation:

$$Parameter_k = C^{n_{\text{best}}}_{\text{output}_k}(p_{\text{best}} + 1), \quad \forall k = 1, \ldots, K.$$

4.4. Learning

In addition to perceiving the environment, retrieving the best matching case, and adapting the schema parameters being used by the reactive control module, SINS must also learn by updating its case library based on its current experience. Three types of learning are possible: modification of the associations contained in a case, creation of a new case based on the current experience, and extension of the size of a case to yield associations over larger time windows. Modification of case contents, in turn,

can be of two types: reinforcement of the associations contained in the case based on a successful experience, and exploration of alternative associations based on an unsuccessful experience.

SINS decides which kind of learning to perform using a relative similarity criterion which determines the quality of the best match. The match value of the best case, based on the case similarity metric, is compared with the match values of the case in previous situations in which it was used. If the current match is worse than the mean match value by more than a standard deviation, the case (although still the best match) is considered to be too different from the current situation, since it has been a better match to other situations in the past. In this case, the *create_case* function is invoked to create a new case containing a sequence of associations formed by copying the values of the sequence of associations in the current environmental representation:

$$C^{n_{\text{best}}}_{\text{input}_j}(0) = E^{n_{\text{best}}}_{\text{input}_j}(0), \qquad \forall j = 1, \ldots, J,$$

$$C^{n_{\text{best}}}_{\text{output}_k}(0) = E^{n_{\text{best}}}_{\text{output}_k}(0), \quad \forall k = 1, \ldots, K.$$

If, on the other hand, the best case matches the current situation well, it is likely that the current situation is representative of the class of situations which the case is beginning to converge towards. If the case provides good recommendations for action, its recommendations should be reinforced; if not, its recommendations should be modified. In SINS, collisions with obstacles and lack of movement are undesirable by definition of the navigation task. A set of schema parameters is considered beneficial if using it does not lead to an undesirable outcome. The objective of the learning functions is to improve the accuracy of prediction of the system's cases and, in turn, to discover those schema parameter values that result in environmental situations that are beneficial for the robot.

If the best case recommends a set of schema parameters that are not beneficial to the robot, the *explore_schemas* function is used to modify the case such that it suggests a different set of schema parameters in similar circumstances in the future. Specifically, the output vectors $C^{n_{\text{best}}}_{\text{output}_k}(p_{\text{best}}+1)$ associated with the environment situation following the best match position p_{best} are modified in a random manner since the current values are not useful to the system. The small random changes allow the system to explore the space of possible schema parameters in a controlled manner. These changes are defined by the following formulae:

$$\rho = \min\left(1, \alpha\ \text{collisions} + \beta\frac{\text{max\_velocity} - \text{velocity}}{\text{max\_velocity}}\right),$$

$$C^{n_{\text{best}}}_{\text{output}_k}(p_{\text{best}}+1) = (1-\rho)\,C^{n_{\text{best}}}_{\text{output}_k}(p_{\text{best}}+1) + \rho\,\text{random}\left(\min C^{n_{\text{best}}}_{\text{output}_k}, \max C^{n_{\text{best}}}_{\text{output}_k}\right),$$

$$\forall k = 1, \ldots, K,$$

where ρ is a "reject" value that determines the extent to which the current recommendations should be taken into account when determining the modified values. A value of $\rho = 0$ specifies that the value of the output vector should be left unchanged, and a

value of $\rho = 1$ specifies that the value of output vector should be replaced completely by a new random value in the allowable range. In any given learning cycle, the value of ρ depends on α and β, which represent the importance of avoiding collisions and moving, respectively. For this article, we used $\alpha = 0.5$ and $\beta = 1.0$, but these values could be changed depending on the desired application. Two different "random" selection functions were implemented and evaluated, one using a Boltzmann distribution and one using a best-reward function; see Section 5.7 for details.

If, on the other hand, the schema parameters suggested by the best matching case produce desirable results, the *reinforce_schemas* function is invoked. This function updates the case by making it more like the current environmental situation, so as to produce the same recommendations in similar situations in the future. This reinforcement is done using the following formulae:

$$\forall i = 0, \ldots, p_{\text{best}}$$
$$C^{m_{\text{best}}}_{\text{input}_j}(i) = \frac{\lambda}{i+1}\left(E_{\text{input}_j}(p_{\text{best}} - i) - C^{m_{\text{best}}}_{\text{input}_j}(i)\right), \qquad \forall j = 1, \ldots, J,$$

$$\forall i = 0, \ldots, p_{\text{best}}$$
$$C^{m_{\text{best}}}_{\text{output}_k}(i) = \frac{\lambda}{i+1}\left(E_{\text{output}_k}(p_{\text{best}} - i) - C^{m_{\text{best}}}_{\text{output}_k}(i)\right), \quad \forall k = 1, \ldots, K,$$

where λ determines the learning rate. For this article, we used $\lambda = 0.9$.

Finally, the *extend_case* function extends the sequence of associations contained in a case. The decision to extend a case is also based on a statistical relative similarity criterion. If the case's predictions $C^{m_{\text{best}}}_{\text{input}_j}(p_{\text{best}}+1)$ are similar to the resulting environment situation within a standard deviation from the mean predictive similarity, and the case does not have more associations in the sequence (that is, it cannot provide a next set of schema parameters), then the case is extended by duplicating the last association of the case:

$$C^{m_{\text{best}}}_{\text{input}_j}(p_{\text{best}} + 2) = C^{m_{\text{best}}}_{\text{input}_j}(p_{\text{best}} + 1), \qquad \forall j = 1, \ldots, J,$$
$$C^{m_{\text{best}}}_{\text{output}_k}(p_{\text{best}} + 2) = C^{m_{\text{best}}}_{\text{output}_k}(p_{\text{best}} + 1), \quad \forall k = 1, \ldots, K.$$

The net result of these learning procedures is to cause the cases in the system's case library to converge towards regularities in the system's interactions with its environment. The system learns useful sequences of schema parameters for different environment situations; these are used to guide navigation and, in turn, are updated based on navigational outcomes so as to improve the reliability of their predictions in similar situations in the future.

5. Evaluation

SINS is fully implemented in C++ on top of Arkin's [3] AuRA architecture for schema-based reactive control. The system works in two modes: real and simulation. In real mode, the system controls a Denning MRV-III robot through a radio link; the

perceptual system of the robot consists of a Denning sonar ring which has twenty-four laboratory grade Polaroid Ultrasonic Rangefinders equally spaced over 360 degrees in a plane parallel to the floor. In simulation mode, the system simulates the values of the Rangefinders and the dynamics of the actual robot.

The methods presented above have been evaluated using extensive experiments across a variety of different types of environments, performance criteria, and system configurations. The results presented here are based largely on experiments performed with the system running in simulation mode, since that allows us to run several hundred experiments in which design parameters and environmental configurations are systematically varied. We measured the qualitative and quantitative improvement in the navigation performance of the SINS system and systematically evaluated the effects of various design decisions on the performance of the system. The results show the efficacy of the methods across a wide range of qualitative metrics, such as flexibility of the system and ability to deal with difficult environmental configurations, and quantitative metrics that measure performance, such as the number of navigational problems solved successfully and the optimality of the paths found for these problems. In addition to the simulation experiments, we also evaluated the system's performance in real mode to verify the validity of our approach.

Evaluation methods must not only verify the performance of the computer system but also provide insight into the range of problems that the system can handle and into the theory that underlies the design of the system. The proposed theory is based on several ideas, some of which have proven successful in past applications of case-based reasoning and multistrategy learning and some of which are introduced in this application. However, formal theoretical models for multistrategy combinations of these methods are not known, and there is no a priori theoretical guarantee of the success of our approach, nor a theoretical basis for predicting or understanding the behavior of the system. This makes it all the important to relate empirical evaluations of the system back to the theory—in other words, to evaluate, not just the program, but the theory underlying the implementation [1, 12, 25].

SINS, like many other case-based reasoning systems, is complex and the domain it operates in is also complex. One result of this is that the behavior of SINS has many sources for variability which cause any performance measure defined to evaluate this behavior to have variability as well. This in turn makes it difficult to assess the significance of an observed behavior of the system in a specific situation. Straightforward performance curves that show how the performance of a system improves over time are not good enough: although these curves show that the performance improves on the specific test problems, they do not provide useful information about why the system works or how the design decisions affect the behavior of the system, nor can they be used to predict the behavior of the system under different circumstances (especially given the variability in the behavior of the system). Ablation studies can be used to analyze the impact of different system's modules in the performance of the system [11, 25]. In such studies, one or more system modules are removed or deactivated to analyze how the performance of the system changes. Although these studies do provide some information about the merit of different modules in the performance of the system, they are based on extreme operating conditions that are often impractical (i.e., one or

more modules are set to be either active or inactive). Moreover, design decisions often deal with allocating certain amount of resources to different modules. Due to their nature, ablation studies can only deal with all-or-nothing resource allocation, disabling the possibility of deciding what would be the optimal amount of resources to allocate to each module.

We used a systematic statistical evaluation methodology (proposed by [52]) to evaluate the performance of the SINS system. In this methodology, statistical tools are used to analyze the change in the performance of the system in terms of changes in design decisions and domain (or problem) characteristics. In such an analysis, the system is evaluated through systematic experiments designed to filter out undesirable sources of variability. The results of the experiments are then analyzed using statistical tools to construct a mathematical model of the behavior of the system in terms of design parameters and domain characteristics. This model is used to assess the merit of various design decisions in the continuous case-based reasoning algorithm and to analyze the performance of the system when it faces problem domains with different characteristics. Such an evaluation enables us to understand the behavior of the system in terms of the theory and design of the computational model, select the best system configuration for a given domain, and predict how the system will behave in case the characteristics of the domain change.

An important characteristic of case-based systems is that they use previous cases to solve new similar problems. Thus, it is important to verify not only that the performance of the system improves, but that the variability in the performance of the system when solving new similar problems decreases as the number of problems solved in the past increases. Any significant indication that the system does not show these behaviors deserves further study since it might reveal possible sources of the utility problem [17,38]. Systematic empirical analysis based on statistical tools can also be used to verify the significance of these behaviors and to identify the sources of variability in the behavior of the system.

5.1. Systematic evaluation of SINS

The performance of SINS varies across different worlds, system configurations, and amounts of experience. Moreover, due to the nature of the task and the architecture of the system, SINS can perform differently given the same world and case library. The reason for this is that the adaptation and learning module tune the navigation module randomly when no appropriate case exists; this allows the system to explore and discover new regularities. This means that any performance metric used to evaluate the system must be treated as a random variable and statistical estimation techniques should be used to assess its mean value.

The next subsections describe in detail the systematic evaluation studies perfomed on SINS. The first set of experiments (study 1) focuses on the effect of two design parameters and one domain characteristic factor on the performance of SINS: maximum number of cases, maximum case size, and world clutter. The first two, maximum number of cases and maximum case size, belong to the design decision group of factors. The third one, world clutter, belongs to the domain (or problem) characteristic group of factors.

We also consider how the experience level influences the performance of SINS and verify that the system indeed improves its performance as the experience level increases.

The second set of experiments (study 2) focuses on how the choice of input representation and adaptation method influence the performance of SINS. Both of these factors represent high-level design decisions that are important in most case-based reasoning systems, including SINS. The input representation refers to the information that the learning and adaptation module of SINS uses to represent and categorize the surrounding environment (situation). The adaptation method refers to the method by which this module adapts the control parameters suggested by the best matching case to the current situation. The experimental procedure for both studies is similar; thus, for the sake of brevity, the systematic evaluation methodology is described in detail only for the first study, and a summary of results is presented for the second study.

The third experiment (study 3) verifies the performance improvement provided by the learning method on an actual robot. We measure the performance of the robot performing in an indoor room with a few static obstacles over several trials. We plot the performance curves against the number of trials to verify that the navigation performance improves along different metrics as the number of trials increases.

5.2. *Study 1: experimental design and data collection*

The objective of both sets of simulation studies is to find an empirical model that describes the relationship between the factors of interest (design parameters and domain characteristics) and system performance as well as the conditions under which such a model is applicable. In this way, it is possible to optimize the performance of the system by selecting the appropriate configuration parameters and to analyze the robustness of the system's performance when operating under conditions that differ from the conditions in which the system was optimized.

To collect data for the evaluation analysis, we performed several runs on the system using a simulation environment that provided a batch mode facility; this facility allowed us to run several hundred simulations to gather statistics on the system's performance over a range of environments. A run consisted of placing the robot at the start location and letting it run until it reached the destination location or until it reached a maximum time limit. The latter condition guarantees termination since some worlds are unsolvable by the system. The data for the estimators was obtained after the system terminated each run. This was to ensure that we were consistently measuring the effect of learning across experiences rather than within a single experience (which is less significant on worlds of this size anyway).

We evaluated the performance of SINS using the median value of the time it takes to solve a world. The reason for this is that the median is a robust estimator of the mean and is not too sensitive to outliers. Outliers are common in schema-based reactive control since the system can get trapped in local minima points, resulting in a significant change in the behavior of the system. An experiment consisted of measuring the time SINS takes to solve a world across five independent runs under the same conditions (i.e., same number of cases, case size, and level of experience, world clutterness) and reporting the median among the five runs as the response variable.

Two experiments were designed to satisfy the objectives of the first study. In the first experiment, we ran different versions of SINS in the same 15% cluttered world, i.e., in a randomly generated world in which 15% of available space is occupied by obstacles. Each system used different configuration parameters. In this way, we collected the data required to build a model that relates the system performance with the configuration parameters and amount of experience when dealing with a specific 15% cluttered world every time. In the second experiment, we ran the best system configuration, as determined by the model created during the first experiment, in a randomly generated 20% cluttered world. In this way, we could verify if the performance of the system holds when the domain characteristic changes.

During both experiments, we collected the data from five independent runs while holding the environmental and system conditions constant. In this way, we could balance out the effects of the configuration parameters and experience level and block out the effects of other parameters such as noise. The first experiment allows us to determine how the design decisions affect system performance (i.e., different systems under the same world or environment). The second experiment allows us to study how different domain characteristics affect system's performance (i.e., the same system under different environments). Further details of the methodology are discussed in [52].

5.3. Model construction

As explained earlier, the performance of SINS is evaluated by estimating the median time to solve a world. Thus, the model that needs to be determined in the first experiment has the median time (T) as the response variable; the model relates T with the configuration parameters and level of experience. We used the following regressors as independent variables: maximum number of cases (C), maximum case size (S), and level of experience (E). We also considered additional regressors such as the quadratic terms C^2, S^2, and E^2 and the quadratic interactions CE, SE, and CS. The reason for considering all these factors is to allow for the possibility that interaction terms may explain variability in the response variable better than individual terms. Statistical analysis was used to reveal which of these terms are really significant and should be considered in the final model. Eq. (1) shows the complete hypothetical model.

$$\begin{aligned} T = {} & \beta_0 + \\ & \beta_C C' + \beta_S S' + \beta_E E' + \\ & \beta_{CS} C' S' + \beta_{CE} C' E' + \beta_{SE} S' E' + \\ & \beta_{CC} C'^2 + \beta_{SS} S'^2 + \beta_{EE} E'^2 + \\ & \varepsilon \end{aligned} \tag{1}$$

where V' is the standardized[8] value of a variable V (i.e., $V' = (V - \bar{V})/\sqrt{var(V)}$).

[8] Use of standardized values instead of the original values helps to reduce roundoff errors and other problems with multicollinearity between independent variables.

Table 1
Best subsets regression results

Number of variables	R^2	R^2_{adj}	$\hat{\sigma}$	C	S	E	CS	CE	SE	C^2	S^2	E^2
1	53.9	53.8	11.031			X						
2	66.7	66.5	9.394	X		X						
3	73.7	73.6	8.346	X		X		X				
4	75.9	75.7	8.002	X		X		X		X		
5	77.5	77.3	7.732	X		X		X		X		X
6	79.0	78.8	7.482	X	X	X		X		X		X
7	79.3	79.0	7.434	X	X	X		X	X	X		X
8	79.6	79.2	7.402	X	X	X		X	X	X	X	X
9	79.8	79.4	7.372	X	X	X	X	X	X	X	X	X

Assuming that the mathematical relationship between the response variable and the independent variables is "smooth", a second order polynomial expression of that relationship, such as the one proposed by the model, is a good approximation. Also, early experiences with the system showed that its behavior was related to the maximum number of allowable cases, maximum case size, and level of experience. The quadratic terms for the maximum number of cases and maximum case size allowed for the possibility of utility problems and the interaction terms were included to allow for the possibility of a direct relationship between the response variable and the terms.[9]

An all-subsets regression analysis was performed to determine which of the terms in the model are really significant (i.e., which terms have influence in the response variable). In this analysis, all possible subsets of regressors are considered and a model is constructed using each subset. Table 1 shows the results of this analysis. We measure the optimality of the model by its R^2_{adj} which is the adjusted coefficient of multiple determination. This coefficient measures the ability of the model to explain changes in the response variable by changes in the regressors. Its range is between 0%, which means that none of the variation in the response is explained by variation in the regressors, and 100% which means that all of the variation in the response is explained by variation in the regressors. Thus, the larger the R^2_{adj} the more explicative is the model.

Table 1 shows the best model obtained within each subset of constant size or number of variables. R^2 shows the coefficient of multiple determination of that model. R^2_{adj} shows the adjusted coefficient of multiple determination.[10] $\hat{\sigma}$ is the estimated standard deviation of the model. The "X" show which variables are included in the best model for each size.

[9] Among the three interaction terms only CS has physical meaning. The interaction term CS is a direct measure of the total amount of memory available to the system. This is an example of a particularly difficult evaluation problem since different design decisions can influence each other under conditions of resource limitations.

[10] The R^2_{adj} is the R^2 adjusted to take into account the number of parameters in the model. This allows models having different number of parameters to be compared.

Table 2
Model coefficients

Coefficients	Value	Std. error	P-value	95% C.I.
β_0	72.23	0.78	0.000	(70.70,73.77)
β_E	−11.92	0.34	0.000	(−12.58, −11.26)
β_C	−5.79	0.34	0.000	(−6.45, −5.13)
β_S	1.97	0.34	0.000	(1.31, 2.63)
β_{EE}	2.33	0.38	0.000	(1.59, 3.07)
β_{CC}	2.99	0.42	0.000	(2.16, 3.82)
β_{SS}	−0.95	0.42	0.024	(−1.78, −0.12)
β_{CE}	−4.32	0.34	0.000	(−4.99, −3.66)
β_{SE}	−0.91	0.34	0.008	(−1.57, −0.24)
β_{CS}	0.74	0.34	0.028	(0.08, 1.41)

Table 3
ANOVA table

Source	df	SS	MS	F	P-value
Regression	9	100675.7	11186.2	205.824	7.1E-157
Residual	470	25543.7	54.3		
Total	479	126219.4			

The best model obtained with the all-subsets analysis corresponds to the one having all nine of the regressors as independent variables[11] (F = 205.824, P-value = 0.000). Table 2 shows the statistical results for each individual parameter in the model as well as the 95% confidence interval estimation of its real value.

Table 3 shows the analysis of variance (ANOVA table). The ANOVA table is a statistical tool that it is used to determine the sources of variability in a model. The first column identifies a source of variation, and the second, third, and fourth columns show the degrees of freedom (df), sum of squares (SS), and mean squared (MS) of a source, respectively. The fifth column shows the value of the F statistic which is used to determine the significance of the regression. The sixth column shows the P-value. A high significance value means that the variation in the response variable is indeed explained by variation of the independent variables or regressors.

Considering this model, the optimal system configuration parameters can be found using standard calculus techniques, i.e., by setting the first partial derivatives of the model with respect the relevant parameters to zero. Eqs. (2) and (3) show the optimal values for C' and S' at a given level of experience E'.

[11] The F statistic is used to determine the significance of the regression. The P-value is the probability determined by F; the lower this value the better the result, since the significance of the regression is (1 − P-value)% which is 100% for P-value = 0.

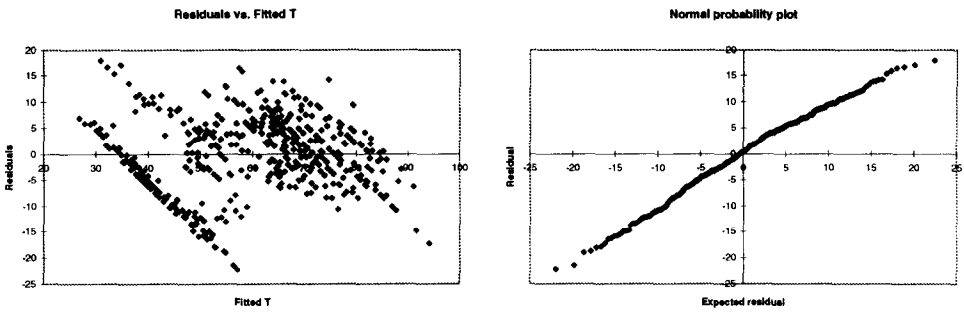

Fig. 5. Residual plots.

$$C' = \frac{2\beta_{SS}\beta_C - \beta_{CS}\beta_S}{\beta_{CS}^2 - 4\beta_{CC}\beta_{SS}} + \frac{2\beta_{SS}\beta_{CE} - \beta_{CS}\beta_{SE}}{\beta_{CS}^2 - 4\beta_{CC}\beta_{SS}} E'$$
$$= 0.80 + 0.75E', \tag{2}$$

$$S' = \frac{2\beta_{SS}\beta_S - \beta_{CS}\beta_C}{\beta_{CS}^2 - 4\beta_{CC}\beta_{SS}} + \frac{2\beta_{SS}\beta_{SE} - \beta_{CS}\beta_{CE}}{\beta_{CS}^2 - 4\beta_{CC}\beta_{SS}} E'$$
$$= 0.05 + 0.41E'. \tag{3}$$

According to these equations the optimal parameter values change with the level of experience. This is due to the interaction terms that exists among those variables. These equations can be used to determine the optimum configuration of the system for a given situation (see Section 5.5).

5.4. Model validation

There are two assumptions that must be verified before accepting the proposed model as a valid model: the residuals have zero mean and constant variance, and the residuals have normal distribution. The least squared error technique relies on these assumptions; since the model coefficients were calculated using this technique we must verify if these assumptions hold. Otherwise, any conclusions derived from the model could be wrong and our understanding of how the factors influence the performance of the system could be misleading. In particular, violation of the assumption about the residuals having zero mean and constant variance could introduce inaccuracy in the estimation of the model coefficients, and violations of the assumption of the residuals being normally distributed could produce underestimation of the confidence intervals (i.e., bigger confidence intervals).

A scatter plot of the residuals against the fitted response was used to diagnose changes in variance and a normal probability plot of the residuals can be used to verify the normality distribution of the residuals. The results of each of these two validation techniques are shown in Fig. 5. The left chart in Fig. 5 shows a constant band of residuals along the horizontal axis. Thus, this chart indicates that the variability of the residuals is constant along the fitted values of the response variable (i.e., median time).

Table 4
Model coefficients

Coefficients	Value	Std. error	P-value	95% C.I.
α_0	80.20	0.71	0.000	(78.57, 81.47)
α_E	−2.86	0.48	0.000	(−3.84, −1.87)
α_{EE}	2.53	0.55	0.000	(1.41, 3.65)

When the variability of the residuals is not constant, the band tends to narrow or widen along the horizontal axis. The right chart in Fig. 5 shows a normal probability plot of the residuals. In this chart, the values of the residuals are plotted against their expected value as drawn from a normal distribution. If the residuals are indeed normal, then the plot should show a straight line that crosses the origin. Since this is actually the case, we can conclude that the residuals are normal.

Since the two assumptions, residuals with zero mean and constant variance and residuals having normal distribution hold, the model can be considered valid.

5.5. Robustness analysis

A second experiment was designed to evaluate the generality of the SINS approach. In this experiment, we evaluated a particular configuration of SINS (the "best" configuration as determined by the previous analysis) performing under different environmental conditions. The data for the experiment was collected in the same manner as the first experiment, the only difference being that the robot solved a fixed randomly generated 20% cluttered world in every run. The configuration parameters for the system were selected using the model constructed in the first experiment and to optimize the performance of the system around an experience level E equal to 20 (i.e., $E' = 0.52$). Subject to these conditions, the system was configured using 43 maximum cases ($C' = 1.19$) of size 11 ($S' = 0.26$).

As in the first experiment, the model that needs to be determined has the median time (T) as the response variable. However, in this case, the model relates the response variable only with the level of experience, since the other factors are constant. In this way, if such a model is found to be significant (i.e., the model shows that the level of experience is related to the response variable), we can conclude that the system still learns under changing environmental conditions. The coefficient derived from this model can be compared with the respective coefficients derived in the previous model. If a significant difference is detected, we can conclude that changing the world clutterness from 15% to 20% affects the learning performance. Eq. (4) shows the complete hypothetical model for the second experiment.

$$T = \alpha_0 + \alpha_E E' + \alpha_{EE} E'^2 + \varepsilon. \tag{4}$$

This model is a simplification of the model in Eq. (1) where only the experience level (E') is included in the regressor. Table 4 shows the statistical results for each individual parameter in the model as well as the 95% confidence interval estimation of its value. As the inferred model shows, a bigger intercept value is obtained which means that the

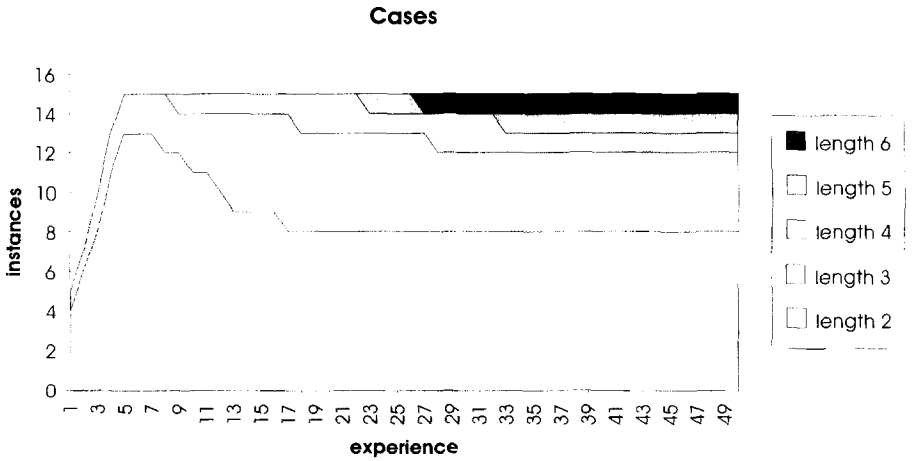

Fig. 6. Case profile.

system indeed needs more time to solve a 20% cluttered world. Also, the increased world clutter has a big influence in the rate of learning (α_E), which is reduced from -17.30 to -2.86. This means that a more experienced level does not improve the performance (reduce the mean time) as fast as in 15% cluttered world. The acceleration of the learning rate (α_{EE}) does not seem to be influenced by the change of world clutter (i.e., it is in the 95% confident interval of β_{EE}).

5.6. Learning profiles

While the above results demonstrate the validity of the SINS approach, it is also interesting to look at the learning profile of the system, if only to provide an intuitive feel for the changes in the internal behaviors and external performance of the system as it gains experience. The learning profile of the system can be determined by assessing the number and the size of the cases in the case library as the experience level of the system increases, and by looking at traditional learning curves based on the system's performance. Fig. 6 shows a graph that displays the number and length of cases against the level of experience for SINS configured as in the previous section. It can be seen that at early stages of experience the system focus on constructing cases with small sequences (i.e., sequences of 2 or 3 samples in the time dimension). As the experience level increases, only those cases containing sequences that have proven to be consistent are progressively extended.

Fig. 6 provides an intuitive understanding of the internals of the continuous case-based reasoning algorithm and the nature of case learning and discrimination process. The externally observable behavior of the system is shown in Fig. 7. The graphs labeled "Median" show the actual time taken by the system (starting with no prior experience or hand-coded high-level knowledge) on a randomly generated world; each point in the graph is the median of five trial runs. The graphs labeled "Fitted T"

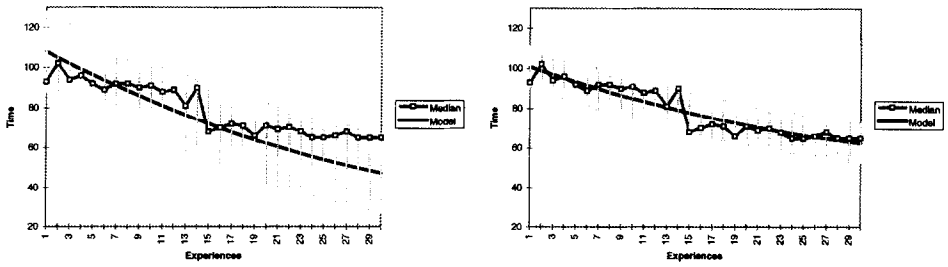

Fig. 7. Performance profiles showing median time from actual system runs and predicted time using "Fitted T" empirical models. On the left is the general model over the entire range of configuration parameters and on the right is the specific model fitted to the configuration parameters used in the system runs.

shows the predicted performance from the empirical model derived in the previous section along with the error bars for that model. The graph on the left describes the performance model over the entire range of configurations of the design parameters; whereas a model for any specific configuration can be found using the same methodology, the observed performance of that configuration should fall within the general model as well. The model on the right describes the performance model for the specific configuration used in these runs; this was obtained by redoing the statistical analysis for the data obtained from this configuration alone. As shown in Fig. 7, the results demonstrate that the system does indeed improve its performance with experience, and that this improvement is as predicted by the empirical models derived from the statistical analysis. Furthermore, the performance of the system approaches its optimal level around $E = 20$, as expected from the design decisions discussed in Section 5.5.

5.7. *Study 2: choice of input representation and adaptation method*

Using the above methodology, we performed a second systematic evaluation study to find an empirical model that describes the relationship between the choice of input representation, adaptation method, and system performance as well as the conditions under which the model is applicable. In this study, we evaluated two choices of input representations and two choices of adaptation methods. The input representation refers to the information that the learning and adaptation module of SINS uses to represent and categorize the surrounding environment (situation). The adaptation method refers to the methods by which this module adapts the control parameters suggested by the best matching case to the current situation.

The reason for studying choices of input representation is to analyze the influence of the "level of information" provided as input on the performance of the system. One option is to use sensory inputs with "low-level" information. In the navigation domain, this might involve using ultrasonic sensors to measure the distance of the closest obstacle in the direction of the goal. Another option is to use sensory inputs that encode "high-level" information; for example, using an array of ultrasonic sensors

to compute a measure of the density of obstacles surrounding the robot. The former type of sensory input is very specific and may allow SINS to discover the best control parameters the robot may use in specific situations. However, we would also expect that the cases learned by the system to be very specific and not work as well in similar but not identical situations. In contrast, the latter type of sensory input is more generic and may allow SINS to discover good control parameters. However, the learned cases may turn so coarse that the system may not perform near-optimally in most cases.

We studied two choices of input representations: the low-level and high-level information types introduced earlier. The low-level information type consists of the following four sensory inputs: *Obstacle-Distance-Ahead*, *Obstacle-Distance-Behind*, *Obstacle-Distance-Right*, and *Obstacle-Distance-Left*. Each of these variables provides a measure of the nearest obstacle that impede navigation in the direction towards, contrary to, right of, and left of the direction of the perceived goal respectively. The high-level information type consists of the following four sensory inputs: *Obstacle-Density* provides a measure of the occupied areas around the robot that impede navigation;[12] *Absolute-Motion* measures the activity of the system; *Relative-Motion* represents the change in motion activity over an appropriate interval; and *Motion-Towards-Goal* specifies how much progress the system has actually made towards the goal. Both types of sensory inputs are computed and constantly updated using the information received from the robot's physical sensors (in our case, 24 ultrasonic sensors arranged around the robot every 15 degrees and shaft encoders).

Another design decision in SINS is the choice of the adaptation algorithm. Every few steps, as determined by a configuration parameter, the adaptation and learning module may recommend new control parameters to the navigation module. To accomplish this, it retrieves the case most similar to the current environmental situation over a recent time window and adapts the control parameters in use by the navigation module based on the values suggested by the case. The best values for the control parameters would be those that increase the likehood of obtaining a positive outcome in the next cycle (as measured by performance metrics such as progress towards the goal or number of collisions with obstacles). In order to learn appropriate control parameters, SINS must "explore" the space of possible parameter values, that is, it must try several values for each environmental situation and learn which values produce positive or negative results. This presents at least two design options for the adaptation method. One option is to select the new values of the control parameters stochastically according to the outcome they have achieved in similar situations in the past. In this method, values that have produced positive outcomes (or obtained positive rewards) are more likely to be selected than those that have produced negative outcomes. Another option is to select for each control parameter the value that has produced the best reward in the past in similar situations. The former adaptation method enforces more exploration and may discover better solutions than the latter one. On the other hand, due to the same reason, the latter method may converge faster than the former. We studied two

[12] Note that this sensory input does not provide any information about the distances or direction of the obstacles; it simply measures the density of occupied area around the robot.

choices of adaptation method: the stochastic method and the select-best method. The stochastic adaptation algorithm selects control parameter values randomly but favoring values that lead to positive reward. Specifically, values are selected according to a Boltzmann distribution: $P(v) = exp(R_v)/\sum_i exp(R_i)$, where the R_i represent the expected reward for value i. The select-best adaptation method simply selects the value that led to the most positive expected reward in the past (i.e., the value with largest R_v).

As in the previous study, we performed experiments to collect data from all possible combinations of design decisions and used it to estimate the parameters of a linear model that relates the performance of the system (i.e., the median time: T) with the independent variables (i.e., the choice of input representation, the choice of adaptation method, and the experience level). We used all-subsets regression analysis to determine which parameters were significant and performed model validation by analyzing the residual plots.

For the purposes of brevity, details of this study are omitted here. The results show that the system configuration that achieves best performance is the one using the high-level representation and the stochastic adaptation algorithm ($T = 51.73$ seconds at $E = 15$ experiences). High-level representations contribute to effective learning independent of the choice of adaptation method. However, when using the stochastic adaptation method the system takes a longer time to converge than when using the best-value adaptation method ($\beta = -12.79$ seconds/experience), although once converged the former provides better performance than the latter. Furthermore, there is no interaction between the choice of input representations and the adaptation procedures that affects learning rate or system performance.

While the results are encouraging, the following limitations should be noted. The normal probability plots show that the residuals do have zero mean, but there is a slight departure from linearity which indicates that the assumption of constant variance and normal distribution is not completely valid. Thus, care must be taken to generalize these results to other worlds. In addition, further research is necessary to determine how well the system transfers its learned knowledge, that is, to investigate the effect of the knowledge acquired in one world on the performance of the system on another world as a function of the choice of input representation, adaptation method, size of case library, and other factors of interest.

5.8. Study 3: experiments with a real robot

While simulation experiments allow us to evaluate the system's performance across a range of design decisions and domain characteristics, it is also important to verify that the system works—and learns—with a real robot. We performed an additional experiment using the actual Denning MRV-III robot to verify the validity of our approach. We used the system configured with the high-level type of input representation and the select-best adaptation method. We ran twenty learning trials and measured the accumulated time and the number of virtual collisions. A trial consisted of placing the robot at the starting point and let it run until either it reached the goal or 300 seconds had elapsed, whichever came first. The experiment was performed in an indoor room with a rectangular free

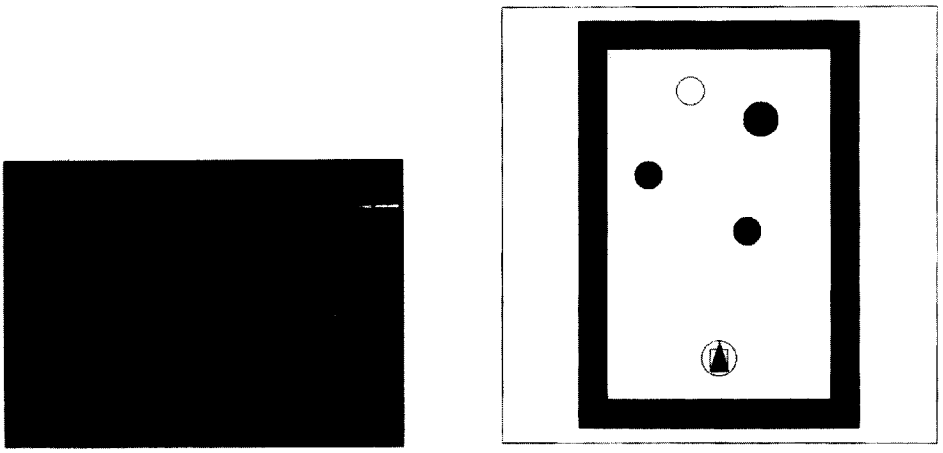

Fig. 8. A picture (left) and a schematic (right) of the robot and its environment. In the schematic, the robot is shown at the bottom of the figure; the hollow circle near the top represents the goal; the black circles represent the static obstacles; and the black bars represent occupied areas that the robot cannot navigate through.

space of approximately 25 × 14 feet and with three circular static obstacles besides the walls. Two of these obstactles were 55-gallon drums with a diameter of 2 feet and the third obstacle was another Denning robot with a diameter of $2\frac{1}{2}$ feet. Fig. 8 shows the robot and the laboratory room used in the experiment. Note that the boundaries of the navigable area are not flat; for example, there are desks and chairs against the walls so that the shape is irregular.

Fig. 9 shows the performance of the robot plotted against the learning trials. The graphs show the time taken (left) and the number of virtual collisions (right) in each trial. During the first two trials the robot was not able to reach the destination point and both trials were manually terminated after five minutes. During the following trials, the robot was able to complete the task successfully and improve its performance. The robot reduced the total time taken from over 5 minutes to a final value of about 2 minutes after the first 10 trials, after which it did not improve further on this metric. Also, it reduced the number of virtual collisions from about 60 to a steady state of about 10 after the first 15 trials, which produced jerky movements along the path. The SINS system, therefore, showed a significant improvement along both metrics. Fig. 10 shows the path taken by the robot in four different trials: 1, 5, 10, and 20.

The next subsection summarizes the results found in the three studies.

5.9. Discussion of results

The performance of SINS is very complex and depends not only on simple terms but also on their interactions. The evaluation shows that the median time the system takes to solve a 15% cluttered world decreases mainly with the experience level. Increasing the maximum number of cases also improves the performance, but a positive coefficient in its quadratic term may deteriorate the performance for big values. On the other

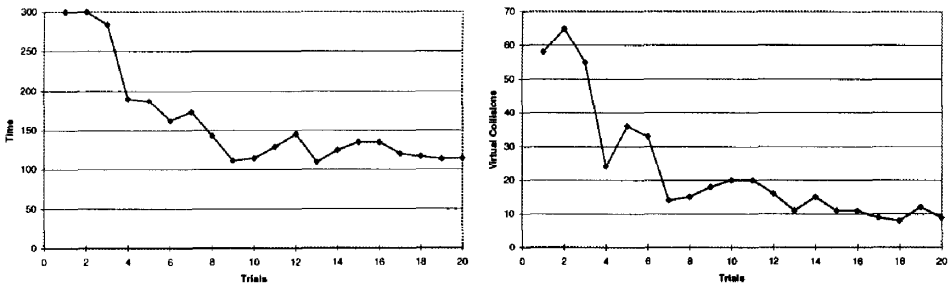

Fig. 9. Performance of the real robot. Left: total time per trial. Right: number of virtual collisions per trial.

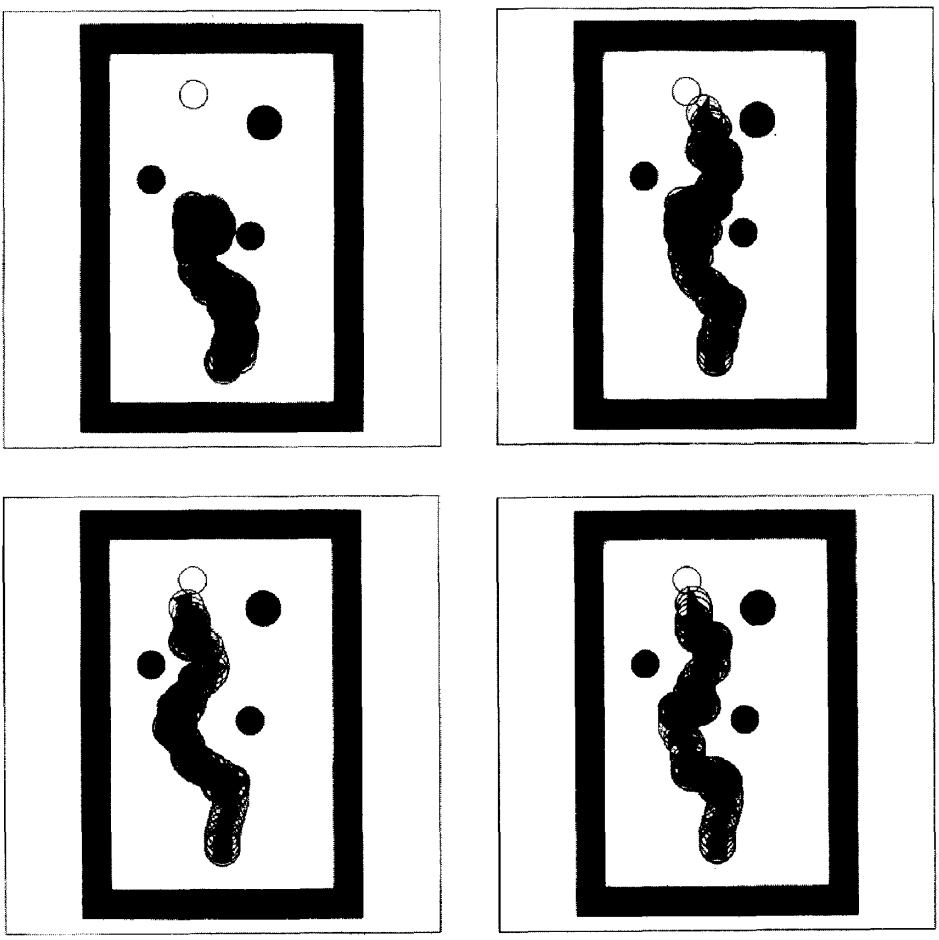

Fig. 10. Actual path followed by the robot in four different trials: trials 1 (top left), 5 (top right), 10 (bottom left), and 20 (bottom right).

hand, the maximum case size has a positive linear coefficient and a negative quadratic coefficient, which indicates that large cases may improve performance as compared to small cases. Negative interaction coefficients indicate that for bigger values of maximum number of cases and case size, the system requires more experiences to start improving its performance. Intuitively, this is to be expected since the more the space available to store regularities, the more the experience required to construct reliable regularities. The performance of SINS is influenced by the world clutter, the learning rate being the factor subject to the greatest influence. Finally, the improvement in performance that is realized is significant in both simulated and real mode.

The results shown above help us verify and understand several aspects of SINS. In particular:

- The evaluation shows that SINS does improve its performance significantly with experience, both in simulation (Table 2 and Fig. 7) and on an actual robot (Fig. 9).
- The performance of the system depends on alternative design decisions, as well as on interactions among them (Eq. (1) and Table 2).
- The choice of input representation and the choice of adaptation method independently influence the performance of SINS and there are no interactions between these decisions (Section 5.7). The model allows us to determine the best choices for these design decisions.
- High-level input representations provide a better starting point for learning than low-level input representations. Also, the stochastic adaptation method takes longer to converge than the best-value adaptation method, but once converged it arrives at better solutions (Section 5.7).
- The model allows us to determine the best way to configure SINS in a 15% cluttered world for a prespecified level of experience (Eqs. (2) and (3)).
- The results show how a change in the environment characteristics, namely clutter, affects the performance of SINS (Eq. (4)).
- The analysis shows that using the proposed factors (C, S, and E) and their interactions the empirical model can only account for 79.8% (i.e., R^2=0.798) of the variability in the performance of the system. Part of the remaining 21.2% could be explained by introducing more factors or by changing the functional forms of the terms in Eq. (1); the rest of the variation in performance is due to the randomness in the system.
- The empirical model in the second study can only account for 65.3% of the variability in the performance of the system. This result is important because it captures the limitations of the derived model. Part of the remaining percentage could be explained by introducing more factors or by changing the functional forms of the terms in the empirical model used.
- When configured with high-level input representations and the select-best adaptation method (as suggested by the simulation studies), the real robot does improve its navigation performance on an indoor navigation task with static obstacles along different performance metrics (Fig. 9).

One interesting and unexpected result deserves mention, one which arose out of the reinforcement learning method that SINS uses for case adaptation. As discussed earlier, we had hypothesized that an internal reward signal based on consistency would

be sufficient to guide learning. This hypothesis turned out to be correct; SINS indeed learns perception-action models which provide accurate predictions about the results of its actions even without an external reward signal. However, these models do not necessarily enhance performance on the navigation task. For example, in our initial experiments, SINS reduced all schema gains to zero; it learned (correctly) that if it turned off all its motor control schemas, it would consistently observe the result that it went nowhere. While this is not useful for the particular task of moving to a destination point, it is nevertheless a "correct" model of action in the physical world and might be useful in other situations (for example, if it needed to stop momentarily to avoid a moving obstacle). As currently implemented, SINS can be run with or without an external reward signal, depending on the user's goals in using the system.

The above evaluations focussed on the SINS system and the continuous case-based reasoning method in particular. However, we believe that the systematic evaluation methodology used here can be successfully and usefully applied to other machine learning and case-based reasoning systems as well. Furthermore, the continuous case-based reasoning method provides several implications for case-based reasoning theories in general; these are discussed next.

6. Discussion and implications

Continuous case-based reasoning is a variation of traditional case-based reasoning that can be used to perform continuous tasks. The underlying steps in the method are similar, namely, situation assessment, case retrieval, adaptation and execution, and learning. The ACBARR and SINS systems perform a kind of case-based planning, and in that respect are similar to CHEF [21]. However, there are several interesting differences due to the continuous nature of the domain, and to the on-line nature of the performance and learning tasks. Our approach is also similar to Kopeikina, Brandau, and Lemmon's [28] use of case-based reasoning for real-time control. Their system, though not intended for robotics, is designed to handle the special issues of time constrained processing and the need to represent cases that evolve over time. They suggest a system that performs the learning task in batch mode during off peak hours. In contrast, our approach combines the learning capabilities of case-based reasoning with the on-line, real-time aspects of reactive control. In this respect, our research is also different from earlier attempts to combine reactive control with other types of higher-level reasoning systems (e.g., [10,39]), which typically require the system to "stop and think". In our systems, the perception-action task and the adaptation-learning task are integrated in a tightly knit cycle, similar to the "anytime learning" approach of Grefenstette and Ramsey [20].

Our approach to continuous case-based reasoning introduces several innovations to the basic case-based reasoning paradigm. Due to the similarity in the assumptions and methods, we conjecture that many of these innovations would be useful in traditional case-based reasoning systems as well. Let us discuss the underlying case-based reasoning issues in more detail.

6.1. Assumptions underlying case-based reasoning

Our approach combines continuous representations, continuous performance, and continuous learning into a single integrated framework. There are three basic assumptions that underlie this approach. First, the interaction between the reasoning system and the environment is causal and consistent. By *causal* we mean that the same actions executed under the same environmental conditions would result in the same outcomes (or similar outcomes, but such variations are much slower than the execution cycle of the system). By *consistent* we mean that small changes in the executed actions under the same environmental conditions would result in small changes in the outcomes. This guarantees that the system can use past experience to guide performance in similar situations in the future and hope to obtain the same results from its actions. The second assumption underlying our approach is that the system's experiences are likely to be *typical* of future experiences, and that the system will usually encounter problem situations that are similar to those that it already has experience with. These assumptions are also common to many traditional case-based reasoning systems (see, for example, the "typicality assumption" of [45]), although they are not always stated explicitly or in this exact manner. The third assumption underlying our approach is discussed next.

6.2. Fine-grained representations

The third assumption is, perhaps, unique to continuous case-based reasoning.[13] In our current work, we have assumed that the problem domain can be represented quantitatively. This is required by the semantic concepts and operations in our method. In particular, our method requires a formal and well-defined similarity metric to judge the similarity between two given situations, which is used to determine the direction and magnitude of the necessary adaptations. While case-based reasoning in non-continuous domains also requires a similarity metric for partial matching, most such metrics used in existing systems are not fine grained enough to determine the degree of similarity between continuous cases or to place situations that could vary continuously and infinitesimally from each other on a similarity scale. This assumption could be relaxed if adequate symbolic representations and similarity metrics could be developed (see, for example, [13], for an approach to continuous planning based on qualitative process theory), but more research is needed into this issue. Furthermore, we require a similarity metric that can compare not just instantaneous descriptions of situations but representations of variations in situations over periods of time.

The fine-grained nature of the representations is important in case adaptation as well. For example, in the SWALE system [24,54], an explanation pattern may be adapted using a rule that recommends that a "man" be substituted by a "horse". Similarly, CHEF

[13] However, note that even domains that have traditionally been treated using symbolic representations, such as medical domains, may require reasoning with continuous representations. For example, Berger's [8] ROENTGEN system, which is used to used design radiation therapy plans for patients, uses representations of continuous attributes. Other domains, such as weather prediction, may also require hybrid symbolic continuous representations; see, for example, [22].

[21] might substitute "chicken" for "beef" in a recipe. But neither system can interpolate between the symbolic representations of the two concepts; SWALE, for example, can not invent a "man-horse" that would be part-way between its representations of "man" and "horse". The continuous representations used in SINS permit arbitrarily fine-grained modifications (within the limits of the computer implementation), providing considerable power to the system. This assumption, too, could be relaxed if adequate algorithms for constructive concept creation, interpolation, and modification were to be developed (see, for example, [40], but currently this ability of the SINS system is outside the scope of traditional "discrete" case-based reasoning systems.

6.3. Time history representations

Our methods represent a novel approach to case-based reasoning for a new kind of task, one that requires continuous, on-line performance. The system must continuously evaluate its performance, and continue to adapt the solution or seek a new one based on the current environment. Furthermore, this evaluation must be done using the simple perceptual features available to a reactive control system, unlike the complex thematic features or abstract world models used to retrieve cases and adaptation strategies in many case-based reasoning systems. Case-based reasoning in such task domains requires a continuous representation of cases, in terms of the available features, that represent the time course of these features over suitably chosen time windows. Relevant cases are retrieved on the basis of a similarity metric that compares the recent history of the current situation with the cases in the case library; cases are adapted, and new cases learned, on the basis of the system's experiences. Additionally, as is desirable in any case-based reasoning system, case representations in SINS are learned and modified continuously while simultaneously being used to guide action.

To our knowledge, SINS is the first AI system to learn time history representations; this ability is a significant improvement over previous case-based reasoning or inductive learning systems. Whereas some case-based reasoning systems, such as CHEF [21] or PRODIGY [60], do represent sequences of (discrete) events and states in their cases, they do not use the recent history of the current situation as an index to these cases. Because the retrieve-adapt-apply cycle in these systems is not integrated with the perceive-act loop, the progression of events in the latter is not used in the former for retrieval or temporal clustering. Similarly, conceptual clustering systems such as COBWEB [16] classify input examples based on feature lists but not on descriptions of how these features vary over time. In effect, SINS can be thought of as learning associative or classificatory rules in which an antecedent is not merely a description of a situation but of the recent history of a situation.

6.4. Abstract cases

In a traditional, symbolic task domain, a case represents an actual experience or an abstraction of one. For example, cases in CHEF [21] represent actual recipes created by the program. In a continuous domain, however, an actual experience consists of the time histories of real parametric values of perceptual and control parameters. For example,

a navigational experience might involve some starting location on a two-dimensional grid, and a destination location of (6.71m, 10.98m) in propioceptive coordinates on that same grid. The experience might consist of the values of perceptual parameters such as the current position of the goal with respect to the robot (an x, y coordinate pair of real numbers), number of obstacles sensed in the immediate vicinity of the robot (an integer), or the distance in meters to the nearest obstacle, and control parameters such as minimum safe distance of approach to an obstacle (in meters), the speed of the robot (in meters per second), or the current heading of the robot (in radians). These parameters vary continuously over time; thus the complete experience is represented by time graphs of each of these parameters. Clearly, it is not feasible to store the entire time history of each of these parameters for each problem that the robot solves, nor is it useful to do so. [14]

Thus SINS learns and stores some abstraction of the actual experience. One might argue that CHEF actually does the same, since an actual cooking experience would involve perceptual input (such as looking at the frying pan and judging whether the ingredients have browned enough) and control output (such as moving the spatula to stir the ingredients in the pot). In CHEF, the experiences have already been abstracted by the programmer and represented in symbolic form. One of the open issues in our research is the automatic extraction of such symbolic representations from the actual continuous experiences of the system (e.g., [30,44]).

6.5. *Virtual cases*

A related, but different, problem with continuous task domains is that an experience can differ from another in infinitely many ways, and differences can range from significant to infinitesimal. Only a tiny fraction of all possible experiences will actually be undergone by the system. However, the power of a case-based reasoning system comes from its cases, and so it is desirable to have a representative library of cases that will cover the range of experiences the system is likely to encounter. We introduce the idea of a *virtual case*, which represents a representative experience that the system could well have had but may or may not actually have had. Rather than trying to remember all the details (down to the grain size defined by the programmer) of each experience, or some abstraction of these details, SINS combines past cases and present experiences to create a virtual experience. This is similar to the abstraction process described earlier, with the difference that a virtual case does not represent an abstract or generalized description of an actual experience but rather a virtual experience derived from a combination of several actual experiences. A virtual case can be thought of as an "average" or "prototypical" experience—a centroid in a region that represents all the points in the space of experiences that are close enough to the centroid.

One problem with virtual cases is that usually there isn't enough a priori information about where such centroids should be located or how big the regions should be; thus, region centroids and boundaries are learned with experience. Since experiences are

[14] However, if the range of allowable variations of perceptual and control parameters is bounded by the nature of the task, such a "memory-based approach" may be in fact be feasible [6].

undergone sequentially, cases must be learned and adapted progressively in an on-going manner. This means the content of a case depends not only on a past experience, but also on alterations introduced by subsequent similar experiences. This approach deviates from standard case-based reasoning where each case contains a previous experience or a generalization of a previous one and future similar experiences do not modify the content of a case (but cf. [45]).

The notion of a virtual case might be useful in symbolic case-based reasoning systems as well. To take a simple example, AQUA learns about and updates existing cases based on new experiences [45]. This process results in hypotheses that may or may not be "true" of a single experience, but are useful and plausible. If used in future reasoning, these hypotheses could be viewed as virtual cases. SINS's virtual cases are more sophisticated since they are continuously refined through use; the refinement algorithms are also different, as discussed below.

6.6. Dynamic memory

The learning algorithms in SINS are designed to make cases "converge" to useful regularities over time. Since this learning process requires on-going problem solving experiences, the result is that the system's memory is used to guide problem solving at the same time as it is being organized into cases having several degrees of reliability and usefulness. Cases that have not been used often may contain actual navigational experiences and may or may not be useful in solving future problems. Cases that have been used more often may contain virtual experiences since they are exposed to the refinement mechanisms that guide cases towards consistent and useful sequences of associations. These cases are usually more reliable in solving future problems.

Thus, the representations and methods used in SINS allow it to incorporate successive similar experiences into a single representation in a gradual and incremental manner, and to differentiate these representations relative to each other. This has two implications. First, SINS carries out a process of constructive conceptual change that consists of using an original set of low-level, sensorimotor concepts to create higher-level, strategic concepts that are new and useful to the system [44]. Second, SINS implements a kind of dynamic memory [53] in which aggregate memory structures (as in Schank's MOPs), are formed to organize experiences that are consistent with each other (as in Schank's example of visits to a dentist and to a doctor). The virtual cases in SINS can be thought of as dynamic reconstructions of prototypical experiences. Part of the learning process is to organize memory into concepts that are relevant and useful to the task at hand, and to use these concepts to interpret the environment in terms of the internally constructed aggregates and to use them to act successfully; this is a fundamental tenet of dynamic memory and a basis for most case-based reasoning systems.

6.7. Two types of behavior modification

As discussed earlier, our systems use case-based reasoning to suggest global modifications (behavior selection) as well as to suggest more local modifications (behavior adaptation). The knowledge required for both kinds of suggestions are stored in a case,

in contrast with traditional case-based reasoning systems in which cases are used only to suggest solutions, and a separate library of adaptation rules is used to adapt a solution to fit the current problem. In many problem domains, even non-continuous ones, planning (or other types of problem solving) cannot be performed separately from plan execution [36]. In such situations, case-based reasoning can be used to propose a plan (or solution) as well as continuously refine it during execution. Another difference of interest is that cases in ACBARR and SINS propose modifications, not directly of the plan or trajectory, but of the reactive control module itself which then result in modifications to the proposed trajectories.

6.8. *Two types of adaptation*

Traditional case-based reasoning systems retrieve cases and adapt the solutions proposed by those cases in order to provide new solutions to new problems at hand. However, in order to build virtual cases, our systems also need to adapt the cases themselves in response to new experiences. This is similar to the incremental case modification process in AQUA [45] in that, in addition to using a case to deal with a new situation, the system can use an experience to learn more about its existing cases.

In our problem domain, cases represent environmental regularities that have been identified by the system through its experience. These provide the predictive power necessary to navigate in future situations. Cases are used for behavior adaptation; in standard case-based reasoning terms, this can be viewed as the process of using the recommendations provided by the case to adapt the solutions currently being pursued by the system. In addition, cases themselves can be adapted through experience; this can be viewed as a process of discovery in which the system develops a model of the world around it. The system "explores" the search space as its case representations traverse this space and find good "niches" representing regularities. The former process could be viewed as a process of "solution adaptation", and the latter as one of "case modification".

The particular method of case modification used in our system is similar to that of Sutton [56], whose system uses a trial-and-error reinforcement learning strategy to develop a world model and to plan optimal routes using the evolving world model, although there are differences as discussed earlier. In general, we hypothesize that case modification may be useful in other types of case-based reasoning systems as well, although different methods may need to be developed to perform the modifications. AQUA's incremental case modification, for example, can be viewed as such an extension to SWALE's solution adaptation [24,54].

Different criteria may also need to be developed for deciding when to modify a case to fit the new experience, when to learn a new case to represent the new experience, and when to use the case for solution adaptation but without modifying it. Our system uses a relative similarity measure to identify potential regularities; the intuition behind this approach was discussed earlier. We believe that a similar approach can be used in other case-based reasoning systems as well to determine whether a new experience is different enough to merit its own case or whether it should be used to modify the information represented in the best available case.

6.9. On-line real-time response

Unlike traditional case-based reasoning systems which rely on deep reasoning and analysis (e.g., [21]), and unlike other machine learning augmentations to reactive control systems which fall back on non-reactive reasoning (e.g., [10]), our method allows the system to continue to perform reactively with very little performance overhead as compared to a "pure" reactive control system. Even if real-time response is not required, however, continuous case-based reasoning could still be used in problem domains which are inherently continuous and require continuous representations.

6.10. Adaptive reactive control and robot navigation

Our research also contributes to reactive control for autonomous robots in the following ways. One, we propose a method for the use of assemblages of behaviors tailored to particular environmental demands, rather than of single or multiple independent behaviors. Two, our systems can select and adapt these behaviors dynamically without relying on the user to manually program the correct behavioral parameters for each navigation problem. Three, the knowledge required for behavior selection and modification is automatically acquired through experience using multiple learning methods. Finally, our system exhibits considerable flexibility over multiple domains. For example, it performs well in uncluttered worlds, highly cluttered worlds, worlds with box canyons, and so on, without any reconfiguration; furthermore, the results of learning in one kind of environment transfer well to other environments. In this article, we have focussed on the case-based reasoning aspects of our work; robot control issues are discussed in [47,48].

Our research also contributes to and extends related work on behavior coordination. For example, in Mataric's [35] approach, the robot learns a topological map of landmarks in a terrain in order to use it for future planning. The robot performs better with experience because it learns the spatial arrangements of landmarks in the terrain and then uses that knowledge to activate appropriate modules to get from the current position to the goal. However, the mappings between conditions and module activations are precompiled at design time. In contrast, in our method the robot learns both the conditions under which to change control parameters and which control parameters are suitable under those conditions. However, our approach does not allow the robot to learn a map or landmark information; the conditions under which module activation information is indexed are descriptive of the robot's immediate surroundings as perceived by the robot.

More research is needed to combine map learning approaches with navigational strategy learning approaches such as ours. Although maps can help in path planning, they are specific to the terrain in which the robot is trained and must be relearned if the terrain changes. In contrast, although navigational strategies do not provide higher-level guidance for path planning, they are robust across terrains. For example, one of the strategies a robot might learn is that when obstacle density starts increasing the robot should slow down and navigate more carefully to avoid a collision. This information is not specific to the terrain in which it is learned. On the other hand, if the terrain

information is indeed available, a landmark-based approach might be able to avoid the obstacle cluster altogether. One approach to combining these techniques might be to use maps (and map learning techniques) to provide larger-scale navigational guidance, and local navigational strategies (and associated learning techniques) to perform the actual navigation once a global route has been planned.

Other approaches have focused on learning the mapping between situations and coordination of behaviors. For example, Maes and Brooks [34] use experiences to update correlations between modules, situations, and external rewards. Using this information, the robot selects only the behaviors that have proven to be positively correlated with positive feedback (or negatively correlated with negative feedback) with past situations that match the current situation. This approach shares much in common with ours, as do other approaches in which reinforcement learning and similar techniques are used to learn useful behavior through reward feedback. One difference between our approach and that of Maes and Brooks is that in their approach the robot may not be able to disambiguate between different situations that produce the same perceptual sensation since there is no history of the recent past. In our approach, experiences are used to create cases that capture regularities between conditions and control parameters over a time window. This allows the system to select the best control parameters based not only on the current input but on the recent history of inputs which provides more information with which to learn and select appropriate control parameters.

7. Conclusions

We have presented a novel method for augmenting the performance of a reactive control system that combines case-based reasoning for on-line parameter adaptation and reinforcement learning for on-line case learning and adaptation. The method is fully implemented and has been evaluated through extensive simulations, rigorous statistical analysis, and actual implementation on a robot.

The power of the method derives from its ability to capture common environmental configurations and regularities in the interaction between the environment and the system through an on-line, adaptive process. The method adds considerably to the performance and flexibility of the underlying reactive control system because it allows the system to select and utilize different behaviors (i.e., different sets of schema parameter values) as appropriate for the particular situation at hand; furthermore, the knowledge required to perform these functions is acquired automatically by the system through experience. SINS can be characterized as performing a kind of constructive representational change in which it constructs higher-level representations (cases) of system-environment interactions from low-level sensorimotor representations [44].

In SINS, the perception-action task and the adaptation-learning task are integrated in a tightly knit cycle, similar to the "anytime learning" approach of Grefenstette and Ramsey [20]. Perception and action are required so that the system can explore its environment and detect regularities; they also, of course, form the basis of the underlying performance task, that of navigation. Adaptation and learning are required to generalize these regularities and provide predictive suggestions based on prior expe-

rience. Both tasks occur simultaneously, progressively improving the performance of the system while allowing it to carry out its performance task in a continuous, on-line manner.

There are still several unresolved issues in our research. While we have been able to determine appropriate time windows for SINS through simulation studies, the size or extent of the cases needed to represent extended experiences in continuous domains is still an open issue [27]. SINS can adjust the extent of its cases dynamically, but although the method appears to work on the problems on which it has been tested, the systematic analysis presented here did not focus on that component of SINS and more research is needed to determine the generality of that aspect of the algorithm. Furthermore, the retrieval process is very expensive and limits the number of cases that the system can handle without deteriorating the overall navigational performance, leading to a kind of utility problem [17,38]. Our current solution to this problem is to place an upper bound on the number of cases allowed in the system. A better solution would be to develop a method for organization of cases in memory; however, conventional memory organization schemes used in case-based reasoning systems (see [27]) assume structured, nominal information (e.g., [14]) rather than continuous, time varying, analog information of the kind used in our cases.

Another open issue is that of the nature of the regularities captured in the system's cases. While SINS' cases do enhance its performance, they are not easy to interpret. Interpretation is desirable, not only for the purpose of obtaining a deeper understanding of the methods, but also for possible integration of higher-level reasoning and learning methods into the system. For example, instead of guessing initial schema parameter values or modifying them incrementally through trial and error, an explanation-based module working on top of the case adaptation module could provide better suggestions for these values, thus speeding up the search process of finding the best schema parameter values associated with a particular environment situation. This requires a symbolic understanding of the knowledge represented in the system's cases.

Despite these limitations, SINS is a complete and autonomous self-improving navigation system, which can interact with its environment without user input and without any pre-programmed "domain knowledge" other than that implicit in its reactive control schemas. As it performs its task, it builds a library of experiences that help it enhance its performance. Since the system is always learning, it can cope with major environmental changes as well as fine tune its navigation module in static and specific environment situations.

Acknowledgements

This research was supported by the Army Research Laboratory and by the Georgia Institute of Technology. We thank Ron Arkin, Kenny Moorman, and Russ Clark for their help with the ACBARR system and for their many useful suggestions during the entire course of this research, and Anthony Francis, Mark Devaney, and anonymous reviewers for their helpful comments on earlier drafts of this paper.

References

[1] D.W. Aha, Generalizing from case studies: a case study, in: *Proceedings Ninth International Conference of Machine Learning*, Aberdeen (1992) 1–10.

[2] P. Agre and D. Chapman, Pengi: an implementation of a theory of activity, in: *Proceedings AAAI-87*, Seattle, WA (1987) 268–272.

[3] R.C. Arkin, Motor schema-based mobile robot navigation, *Int. J. Rob. Res.* **8** (4) (1989) 92–112.

[4] R.C. Arkin and D.C. MacKenzie, Temporal coordination of perceptual algorithms for mobile robot navigation, *IEEE Trans. Rob. Autom.* **10** (1994) 276–286.

[5] K. Ashley and E. Rissland, Compare and contrast, a test of expertise, in: *Proceedings AAAI-87*, Seattle, WA (1987) 273–284.

[6] C.G. Atkeson, Memory-based learning in intelligent control systems, in: *Proceedings 1990 American Control Conference*, San Diego, CA (1990) 988.

[7] T. Balch, G. Boone, T. Collins, H. Forbes, D. MacKenzie and J.C. Santamaría, Io, Ganymede, and Callisto: a multiagent robot janitorial team, *AI Magazine* **16** (2) (1995) 39–51.

[8] J. Berger, ROENTGEN: radiation therapy and case-based reasoning, in: *Proceedings Tenth Conference on Artificial Intelligence for Applications*, San Antonio, TX (1994) 171–177.

[9] R. Brooks, A robust layered control system for a mobile robot, *IEEE J. Rob. Autom.* **2** (1986) 14–23.

[10] S.A. Chien, M.T. Gervasio and G.F. DeJong, On becoming decreasingly reactive: learning to deliberate minimally, in: *Proceedings Eighth International Workshop on Machine Learning*, Chicago, IL (1991) 288–292.

[11] P.R. Cohen and A.E. Howe, How evaluation guides AI research, *AI Magazine* **9** (4) (1988) 35–43.

[12] P.R. Cohen and A.E. Howe, Towards AI research methodology: three case studies in evaluation, *IEEE Trans. Syst. Man Cybern.* **19** (1989) 634–646.

[13] G.F. DeJong, Learning to plan in continuous domains, *Artif. Intell.* **65** (1994) 71–141.

[14] E.A. Domeshek, Do the right thing: a component theory for indexing stories as social advice, Ph.D. thesis, Department of Computer Science, Yale University, New Haven, CT (1992).

[15] R.E. Fikes, P.E. Hart and N.J. Nilsson, Learning and executing generalized robot plans, *Artif. Intell.* **3** (1972) 251–288.

[16] D. Fisher, Knowledge acquisition via incremental conceptual clustering, *Mach. Learn.* **2** (1987) 139–172.

[17] A. Francis and A. Ram, Computational models of the utility problem and their application to a utility analysis of case-based reasoning, in: *Proceedings ML-93 Workshop on Knowledge Compilation and Speedup Learning*, Amherst, MA (1993).

[18] M. Georgeff, Planning, *Ann. Rev. Comput. Sci.* **2** (1987) 359–400.

[19] A. Goel, A. Khaled, M. Donnellan, A. Gomez De Silva and T. Callantine, Multistrategy adaptive navigational path planning, *IEEE Expert* **9** (6) (1994) 57–65.

[20] J.J. Grefenstette and C.L. Ramsey, An approach to anytime learning, in: *Proceedings Ninth International Conference on Machine Learning*, Aberdeen (1992) 189–195.

[21] K.J. Hammond, *Case-Based Planning: Viewing Planning as a Memory Task*, Perspectives in Artificial Intelligence (Academic Press, Boston, MA, 1989).

[22] E.K. Jones and A. Roydhouse, Intelligent retrieval of archived meteorological data, *IEEE Expert* **10** (1995) 6.

[23] L. Kaelbling, *Learning in Embedded Systems* (MIT Press, Cambridge, MA, 1993).

[24] A. Kass, Tweaker: adapting old explanations to new situations, in: R.C. Schank, A. Kass and C.K. Riesbeck, eds., *Inside Case-Based Explanation* (Lawrence Erlbaum, Hillsdale, NJ, 1994) 263–295.

[25] D. Kibler and P. Langley, Machine learning as an experimental science, in: *Proceedings Third European Working Session on Learning*, Glasgow (1988) 81–92.

[26] J.L. Kolodner, R.L. Simpson and K. Sycara, A process model of case-based reasoning in problem solving, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985) 284–290.

[27] J.L. Kolodner, *Case-Based Reasoning* (Morgan Kaufmann, San Mateo, CA, 1993).

[28] L. Kopeikina, E. Brandau and A. Lemmon, Case-based reasoning for continuous control, in: *Proceedings Workshop on Case-Based Reasoning*, Clearwater Beach, FL (1988) 250–259.

[29] B.J. Kuipers, A qualitative approach to robot exploration and map learning, in: *Proceedings AAAI Workshop on Spatial Reasoning and Multi-Sensor Fusion*, St. Charles, IL (1987) 390–404.

[30] B.J. Kuipers and Y-T. Byun, A robust, qualitative method for robot spatial learning, in: *Proceedings AAAI-88*, St. Paul, MN (1988) 774–779.

[31] D. Langer, J.K. Rosenblatt and M. Hebert, A behavior-based system for off-road navigation, *IEEE Trans. Rob. Autom.* **10** (1994) 776–783.

[32] D.B. Leake, Learning adaptation strategies by introspective reasoning about memory search, in: *Proceedings AAAI-93 Workshop on Case-Based Reasoning* (AAAI Press, Menlo Park, CA, 1993) 57–63.

[33] P. Maes, Situated agents can have goals, *Rob. Autonom. Syst.* **6** (1990) 49–70.

[34] P. Maes and R.A. Brooks, Learning to coordinate behaviors, in: *Proceedings AAAI-90*, Boston, MA (1990) 796–802.

[35] M.J. Mataric, Environment learning using a distributed representation, in: *Proceedings IEEE Conference on Robotics and Automation* (1990) 402–406.

[36] D. McDermott, Planning and acting, *Cognit. Sci.* **2** (2) (1978) 71–109.

[37] S. Minton, Learning effective search control knowledge: an explanation-based approach, Ph.D. thesis, Tech. Rept. CMU-CS-88-133, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA (1988).

[38] S. Minton, Quantitative results concerning the utility of explanation-based learning, *Artif. Intell.* **42** (1990) 363–391.

[39] T.M. Mitchell, Becoming increasingly reactive, in: *Proceedings AAAI-90*, Boston, MA (1990) 1051–1058.

[40] K. Moorman and A. Ram, A model of creative understanding, *Proceedings AAAI-94*, Seattle, WA (1994) 74–79.

[41] J. Mostow and N. Bhatnagar, FAILSAFE—a floor planner that uses EBG to learn from its failures, in: *Proceedings IJCAI-87*, Milan (1987) 249–255.

[42] S. Pandya and S. Hutchinson, A case-based approach to robot motion planning, in: *Proceedings 1992 IEEE International Conference on Systems, Man, and Cybernetics*, Chicago, IL (1992) 492–497.

[43] D. Payton, An architecture for reflexive autonomous vehicle control, in: *Proceedings IEEE Conference on Robotics and Automation*, San Francisco, CA (1986) 1838–1845.

[44] A. Ram, Creative conceptual change, in: *Proceedings Fifteenth Annual Conference of the Cognitive Science Society*, Boulder, CO (1993) 17–26.

[45] A. Ram, Indexing, elaboration & refinement: incremental learning of explanatory cases, *Mach. Learn.* **10** (1993) 201–248.

[46] A. Ram, R.C. Arkin, G. Boone and M. Pearce, Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation, *Adapt. Behav.* **2** (1994) 277–305.

[47] A. Ram, R.C. Arkin, K. Moorman and R.J. Clark, Case-based reactive navigation: a case-based method for on-line selection and adaptation of reactive control parameters in autonomous robotic systems, Tech. Rept. GIT-CC-92/57, College of Computing, Georgia Institute of Technology, Atlanta, GA (1992).

[48] A. Ram and J.C. Santamaría, Multistrategy learning in reactive control systems for autonomous robotic navigation, *Informatica* **17** (1993) 347–369.

[49] C.K. Riesbeck and R.C. Schank, *Inside Case-Based Reasoning* (Lawrence Erlbaum, Hillsdale, NJ, 1989).

[50] S.J. Rosenschein and L. Kaelbling, The synthesis of digital machines with provable epistemic properties, in: J. Halpern, ed., *Proceedings Conference on Theoretical Aspects of Reasoning about Knowledge*, Monterey, CA (1993).

[51] E.D. Sacerdoti, *A Structure for Plans and Behavior* (Elsevier, New York, 1977).

[52] J.C. Santamaría and A. Ram, Systematic evaluation of design decisions in case-based reasoning systems, in: *Proceedings AAAI-94 Workshop on Case-Based Reasoning*, Seattle, WA (1993) 23–29.

[53] R.C. Schank, *Dynamic Memory: A Theory of Learning in Computers and People* (Cambridge University Press, New York, 1982).

[54] R.C. Schank, *Explanation Patterns: Understanding Mechanically and Creatively* (Lawrence Erlbaum, Hillsdale, NJ, 1986).

[55] A.M. Segre, *Machine Learning of Robot Assembly Plans* (Kluwer Academic Publishers, Norwell, MA, 1988).

[56] R.S. Sutton, Integrated architectures for learning, planning, and reacting based on approximating dynamic programming, in: *Proceedings Seventh International Conference on Machine Learning*, Austin, TX (1990) 216–224.

[57] R.S. Sutton, ed., *Special Issue on Reinforcement Learning*, *Mach. Learn.* **8** (3–4) (1992).

[58] R.S. Sutton, A.G. Barto and R.J. Williams, Reinforcement learning in direct adaptive optimal control, in: *Proceedings American Control Conference*, Boston, MA (1991) 2143–2146.

[59] E.L. Thorndike, *Animal Intelligence* (Hafner, Darien, CT, 1911).

[60] M. Veloso and J.G. Carbonell, Derivational analogy in PRODIGY: automating case generation, storage, and retrieval, *Mach. Learn.* **10** (1993) 249–278.

[61] P.F.M.J. Verschure, B.J.A. Kröse and R. Pfeifer, Distributed adaptive control: the self-organization of structured behavior, *Rob. Autonom. Syst.* **9** (1992) 181–196.

[62] C.J.C.H. Watkins, Learning from delayed rewards, Ph.D. thesis, University of Cambridge, Cambridge (1989).